2012

Quantifying Performance Costs of Database Fine-Grained Access Control

David Harold Kumka
Nova Southeastern University, dave.kumka@dataexperts.ca

Quantifying Performance Costs of Database Fine-Grained Access Control

by
David H. Kumka

A dissertation report submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy
in
Information Systems

Graduate School of Computer and Information Sciences
Nova Southeastern University
2012

An Abstract of the Dissertation Report Submitted to Nova Southeastern University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Quantifying Performance Costs of Database Fine-Grained Access Control

by

David H. Kumka

October 2012

Fine-grained access control is a conceptual approach to addressing database security requirements. In relational database management systems, fine-grained access control refers to access restrictions enforced at the row, column, or cell level. While a number of commercial implementations of database fine-grained access control are available, there are presently no generalized approaches to implementing fine-grained access control for relational database management systems.

Fine-grained access control is potentially a good solution for database professionals and system architects charged with designing database applications that implement granular security or privacy protection features. However, in the oral tradition of the database community, fine-grained access control is spoken of as imposing significant performance penalties, and is therefore best avoided. Regardless, there are current and emerging social, legal, and economic forces that mandate the need for efficient fine-grained access control in relational database management systems.

In the study undertaken, the author was able to quantify the performance costs associated with four common implementations of fine-grained access control for relational database management systems. Security benchmarking was employed as the methodology to quantify performance costs. Synthetic data from the TPC-W benchmark as well as representative data from a real-world application were utilized in the benchmarking process.

A simple graph-base performance model for Fine-grained Access Control Evaluation (FACE) was developed from benchmark data collected during the study. The FACE model is intended for use in predicting throughput and response times for relational database management systems that implement fine-grained access control using one of the common fine-grained access control mechanisms – authorization views, the Hippocratic Database, label-based access control, and transparent query rewrite. The author also addresses the issue of scalability for fine-grained access control mechanisms that were evaluated in the study.

Acknowledgement

I would like to acknowledge the contribution of my committee chair, Dr. Junping Sun for his invaluable guidance and support as well as the contribution of the other members of my committee, Dr.James Cannady and Dr. Marlyn Littman. The knowledge gained from their courses, as well as from other instructors in the doctoral program, helped to prepare me for the work required to undertake this dissertation. Thank you all!

Table of Contents

List of Tables

Tables

List of Figures

Figures

Chapter 1

Introduction

Controlling security in a relational database management system includes managing the confidentiality, integrity, and availability of stored data. Fine-grained access control can be employed with relational database management systems to ensure confidentiality of data at the column, row, and even at the cell level (Zhu, Shi, Wang, & Feng, 2008). However, current approaches to implementing fine-grained access control for relational database management systems incur significant challenges. These challenges can be summarized as follows:

1. Fine-grained access control is a conceptual approach to providing enhanced security and privacy protection – it is not a specific technology. However, according to Wang et al. (2007), existing approaches to providing fine-grained access control for relational database management systems all have known problems. These problems include the return of incorrect query results, the return of incomplete query results, and leakage of information in contravention of fine-grained security policies (Wang et al.).
2. Fine-grained access control has historically been implemented through custom applications external to the relational database management system. However, external fine-grained access control solutions are much less effective than fine-grained access control implemented directly within the database, as

internal solutions generally provide a significantly reduced attack surface (Roichman & Gudes, 2007). Moreover, security and privacy controls are much less likely to be bypassed if fine-grained access control is implemented within the database (Zhan, Li, Ye, & Wang, 2006). Recently, IBM®, Oracle®, and Sybase® have begun to offer fine-grained access control capabilities integrated within their relational database management systems (Kabra, Ramamurthy, & Sudarshan, 2006).

3. The use of fine-grained access control imposes performance penalties. Siegenthaler and Birman (2009) described this issue as the trade-off between the granularity of data protection and database performance. Still, some implementations of fine-grained access control can impose significant performance penalties. For example, Zhan et al. (2006) reported a 20-30% performance penalty when implementing a novel fine-grained access control solution within the PostgreSQL relational database management system.

The access control mechanism provided natively in a relational database management system is known as "coarse-grained" access control (Wang et al., 2007). Coarse-grained access control for database objects is configured through Structured Query Language (SQL) using the statements GRANT and REVOKE to respectively permit or deny access to database objects (Majedi, Ghazinour, Chinaei, & Barker, 2009). However, Chaudhuri, Dutta, and Sudarshan (2007) characterized coarse-grained access control as all or nothing. Specifically, once coarse-grained access on a database table is granted to a database user, all rows in the table become accessible to that user, regardless of the sensitivity of the data in individual rows (Kabra et al., 2006).

In contrast to coarse-grained access control, fine-grained access control allows specific rows, columns, and even cells in a database table to be given different authorizations (Wang et al., 2007). Multiple factors motivate the need for fine-grained access control in relational database management systems. A key motivating factor is the requirement to provide secure access to relational database management systems from the Web (Roichman & Gudes, 2007). Increasing emphasis on data privacy and compliance with new, more encompassing privacy regulations are additional factors motivating the requirement for fine-grained access control (Bertino, Byun, & Li, 2005). The European Union Data Protection Directive and the Graham-Leach-Billey Act in the United States as are two examples of legislation driving the need for better privacy controls in database systems (Johnson & Grandison, 2007). However, the need to secure electronic healthcare data is often given as the leading example of privacy-sensitive information that could benefit significantly from the privacy preserving capabilities provided by fine-grained access control (Siegenthaler & Birman, 2009). In addition, considerable interest has been expressed in providing fine-grained authorizations for social networking applications in order to secure the privacy of personal data (Simpson, 2008; Majedi et al., 2009).

Fine-grained access control enforces the rules specified in an access control policy (Wang et al., 2007). Access control policies may consist of both processes and structures utilized to document the rules for disclosure and use of data (Agrawal, Grandison, Johnson, & Kiernan, 2007). Thus, the definition of an access control policy can be broad and at times ambiguous. For example, Currim, Jung, Xiao, and Jo (2009) described access control policies as collections of rules with each rule being equivalent to a database view. Bruns, Dantas, and Huth (2007) described access control policies as

predicates that precisely specify access to data. Cho, Kim, Hong, and Cho (2009) provided yet another definition for access control policies – access control policies are lists of rules, which may exhibit a defined hierarchy and structure.

Custom programming, using code embedded within individual applications, is the most widely adopted approach to providing fine-grained access control for applications that interface with relational database management systems (Kabra et al., 2006). Nevertheless, custom fine-grained access control solutions can be very difficult to maintain, particularly if a separate custom solution must be deployed for each single application (Wang et al., 2007). In addition, custom fine-grained access control solutions implemented at the application level can be easily bypassed (Rizvi, Mendelzon, Sudarshan, & Roy, 2004). Johnson and Grandison (2007) and Franzoni, Mazzoleni, Valtolina, and Bertino (2007) independently proposed conceptually similar approaches to fine-grained access control using a "middleware" layer. A middleware layer providing fine-grained access control may be shared by multiple applications without the need to modify the underlying application code (Johnson & Grandison, 2007). Still, middleware security solutions, as is the case with application-level security solutions, can be easily circumvented by connecting directly to the database. For example, reporting tools that connect directly to a relational database management system using an Open Database Connector (ODBC) provide a simple means to bypass fine-grained access controls implemented at the application or middleware layer (Santos & Bernardino, 2009).

Implementing fine-grained access control within a relational database management system provides better security than either custom or middleware solutions (Roichman & Gudes, 2007). For example, Zhan et al. (2006) noted that fine-grained

access control implemented within a relational database management system is much more difficult to subvert than a fine-grained access control solution that is implemented outside the relational database management system. Moreover, fine-grained access control implemented at the database level ensures that security for both applications and ad hoc users is applied consistently (Wang et al., 2007). A number of commercial database vendors have responded to the need for fine-grained access control integrated within the database. As a case in point, IBM®, Oracle®, and Sybase® now offer proprietary implementations of fine-grained access control within their respective relational database management products (Kabra et al., 2006). Microsoft® also provides the capability to implement row and cell-level fine-grained access control in the SQL Server® database, although the functionality is more loosely integrated than in the IBM®, Oracle®, or Sybase® products (Zhang, 2008).

Over the last decade, there has been a significant increase in academic research pertaining to fine-grained access control for relational database management systems (Zhu et al., 2008). Still, Zhang (2008) acknowledges that little work has been done "…to address the performance issues of database systems with fine-grained access controls" (p. iii). Yet according to Kabra et al. (2006), performance is a key consideration when implementing fine-grained access control. Zhu and Lü (2007) summarized the current state of fine-grained access control for relational database management systems with the following statement, "Providing efficient and effective fine-grained access control for database management systems has long been an unresolved issue, however it is critically important for Internet-based data management systems" (p. 222). Zhu et al. (2008) cautioned that the integration of Web and database technology exposes large volumes of

valuable data to potential attacks from the Internet. Yu (2009) believes that fine-grained access controls are a critical requirement for Internet accessible Web-database applications "…to ensure the legal use of data and to prevent privacy breach" (p. 230).

Statement of the Problem

The problem that was studied in this dissertation concerns the performance penalties imposed by the use of fine-grained access control in relational database management systems. The work conducted addresses a significant problem that confronts database professionals and system architects when implementing fine-grained access control for relational database management systems. Specifically, which approach to providing fine-grained access control offers better performance? In the study conducted, the author undertook to quantify performance and scalability considerations associated with the use of fine-grained access control for relational database management systems. The results of author's study provide empirical data concerning the performance and scalability for four fine-grained access control mechanisms. "Security in computing, as in anything else, comes with cost and overhead" (Benantar, 2005, p. 6).

Performance penalties associated with the use of fine-grained access control in relational database management systems are known to be a significant concern. Johnson and Grandison (2007) in discussing privacy enforcement using fine-grained access control maintained that, "Effective privacy solutions must also be economically and computationally efficient…" (p. 256). However, much of the past research relating to the performance costs of fine-grained access control is narrowly focused – only the performance characteristics of novel solutions have been evaluated. For instance, LeFevre¸ Agrawal, Ercegovac, Ramakrishnan, Xu, and DeWitt (2004), Kabra et al.

(2006), Zhu and Lü (2007), and Zhu et al. (2008), all demonstrated statistically significant impacts to database performance when employing fine-grained access control within their specific research projects. In other cases, for example, Rizvi et al. (2004), Franzoni et al. (2007), and Pun, Chinaei, and Barker (2009), performance of fine-grained access control mechanisms was identified as an important research consideration, but was left as a task for future investigation.

Dissertation Goal

The goal of the study was to develop a simple generic model to address performance and scalability for fine-grained access control in relational database management systems. Melnik, Rahm, and Bernstein (2003) described the function of a generic model as providing a paradigm to illustrate concepts at a high level of abstraction using concise definitions. In the study conducted, the author employed a widely accepted benchmark to evaluate the performance and scalability of four implementation models of database fine-grained access control. According to Menascé and Almeida (2001), a benchmark must be relevant, understandable, scalable, and most importantly, widely accepted. The Transaction Processing Performance Council (TPC) has published a number of well-documented benchmarks (e.g., TPC-C, TPC-W) that are widely accepted (Menascé and Almeida). TPC benchmarks have been used in a number of previous studies to evaluate the performance impacts of fine-grained access control in relational database management systems (Kabra et al., 2006; Zhan et al., 2006; Zhu & Lü, 2007; Zhu et al., 2008). As well, TPC benchmarks continue to be used in research projects that involve evaluation of database performance and system scalability (Chaudhuri, 2009; He

& Veeraraghavan, 2009; Elnikety, Dropsho, Cecchet, & Zwaenepoel, 2009; Santos & Bernardino, 2009; Chakraborty, Majumdar, & Sural, 2010; Ahmad, Duan, Aboulnaga, & Babu, 2011).

The fine-grained access control models evaluated by the author include authorization views, transparent query rewrite, the Hippocratic Database, and label-based access control (LBAC). Zhu et al. (2008) categorized these four implementation models as being representative of current approaches to providing fine-grained access control for relational database management systems. These four implementation models can be summarized as follows:

1. Rizvi et al. (2004) and Roichman and Gudes (2007) discussed the use of parameterized authorization views for fine-grained access control. According to Kabra et al. (2006), authorization views may be implemented in any relational database management system using standard SQL.
2. Transparent query rewrite provides the foundation for Oracle® row-level security, now called Oracle® Virtual Private Database (Bertino et al., 2005). In the Oracle® Virtual Private Database (VPD), the SQL WHERE clause is rewritten to include predicates from a corresponding fine-grained access control policy (Shi, Zhu, Fu, & Jiang, 2009). Transparent query rewrite uses the database query rewriter (an internal relational database component) to modify user queries in order to return only authorized data (Hellerstein, Stonebraker, & Hamilton, 2007).
3. The Hippocratic Database provides a generic approach to fine-grained access control that can be implemented in most relational database management

systems (Agrawal et al., 2007). The Hippocratic Database is primarily intended to provide enhanced privacy preservation, but also provides fine-grained authorizations (Johnson & Grandison, 2007). An implementation of the Hippocratic Database by LeFevre et al. (2004) employed query modification explicitly specified in SQL statements to enforce privacy policies – the privacy policies themselves were specified in the database. Yu (2009) described the implementation by LeFevre et al. as "…a practical and efficient approach to incorporating privacy policy enforcement into an existing application and database environment…" (p. 330).

4. LBAC is a feature available in commercial relational database products from Oracle® and IBM® and can be emulated in Microsoft's SQL Server® (Zhang, 2008). In LBAC, each row in the database contains a security label (Bertino et al., 2005). LBAC is used in high security environments to secure classified data (Bishop, 2002).

Vieira and Madeira (2005) proposed the use of standard benchmark suites as suitable instruments for evaluating the overhead imposed by database security mechanisms. Vieira and Madeira employed the TPC Benchmark™ W (TPC-W) to evaluate security overhead. Others, including Zhu et al. (2006), Manjhi, Ailamaki, Maggs, Mowry, Olston, and Tomasic (2006), and Zhu and Lü (2007), have employed the TPC-W benchmark to evaluate the performance of various security mechanisms in relational database management systems. However, much of the existing research on the topic of fine-grained access control utilizes benchmarking only as a means to validate

performance characteristics of novel solutions (Zhu & Lü, 2007; Olson, Gunter, & Madhusudan, 2008; Shi et al., 2009; Krishnamurthy, Mettler, & Wagner, 2010).

Small-scale database systems can be used effectively to predict the performance and scalability of larger database systems. Elnikety et al. (2009) demonstrated the feasibility of using the TPC-W database benchmark on small-scale database servers to validate scalability predictions of analytical performance models. The approach taken by Elnikety et al. was to compare the goodness of the fit between predicted performance and actual results obtained from the TPC-W benchmark. The small-scale experimental configuration employed by Elnikety et al. utilized a readily available combination of commodity hardware and software – a single-core x86 architecture CPU, one gigabyte of server RAM, a single 120 GB disk drive, the Linux operating system, and the PostgreSQL open source database.

Based upon the findings of this study, the author developed a simple generic model using benchmark results compiled from the PostgreSQL open source database. The proprietary fine-grained access control solutions marketed by Oracle® and IBM®, respectively the Virtual Private Database and LBAC, were emulated within the PostgreSQL database. As well, authorization views and the Hippocratic Database, which can be implemented within any relational database management system, were also evaluated using the PostgreSQL database. The goal in developing the generic model was threefold:

1. First, the generic model provides a reference viewpoint that encompasses current understanding of the performance issues attendant with the use of fine-grained access control for relational database management systems.

2. Second, the generic model describes performance behavior that can be consistently reproduced using implementations of fine-grained access control mechanisms currently available for relational database management systems.
3. Third, the generic model offers a research-based foundation for future scholarly work. The model that was developed provides performance comparisons of multiple implementation of fine-grained access control using the same server and database environment.

Relevance and Significance

The 2009 Claremont Report on Database Research identified data security and data privacy as among the leading issues facing database researchers today (Agrawal et al., 2009). To clarify, data security is concerned with ensuring the confidentiality, integrity, and availability of data (Bertino & Sandhu, 2005). On the other hand, data privacy is concerned with enforcing the rights of individuals to determine how and when personal information is released (Agrawal, Kiernan, Srikant, & Xu, 2002). According to Yu (2009), current research in the area of database security tends to treat data security and data privacy protection as two completely separate issues. However, Yu found this situation unsatisfactory given requirements "… to provide a mechanism which supports access control and privacy protection in [the] DBMS simultaneously…" (p. 330).

Significantly, a number of existing implementations of fine-grained access control are capable of satisfying requirements for both data security and data privacy. Fine-grained access control for relational database management systems can also provide enhanced data security (Zhu et al., 2008). Enhanced data privacy is possible using

implementations of fine-grained access control that employ privacy policies specified in metadata or privacy-policy tables (Bertino et al., 2005). Protection of electronic healthcare information is a leading example of data that could benefit from mechanisms providing both enhanced data security and enhanced privacy protection.

In a 2004 report, the U.S. President's Information Technology Advisory committee (PITAC) stated that the capability to ensure privacy of health care records is critical in transitioning from paper-based health care records to networked, electronic health care systems (Agrawal et al., 2007). Johnson and Grandison (2007) described the requirement to exchange health care data as an area where fine-grained access control can provide a suitable mechanism for enforcement of data privacy. However, privacy concerns for electronic healthcare systems continue to be a significant challenge to the present day (He & Yang, 2009; Siegenthaler & Birman, 2009)

Fine-grained access control provides a viable mechanism for protecting Web accessible databases. Zhu and Lü (2007) characterized the need for fine-grained access control in large-scale database systems as the logical outcome of the integration between Internet-based technologies and enterprise relational database management systems. Zhu et al. (2008) described the capabilities provided by fine-grained access control as offering access control that is both flexible and effective in Web accessible databases. Pan (2009) stated that fine-grained access control provides a workable mechanism to protect Web accessible databases from malicious attacks. Yu (2009) perceived fine-grained access control for relational database management systems as a mechanism to ensure secure access to data, as well as providing an effective means to preventing privacy breaches.

The requirements for fine-grained access control are not restricted simply to relational database management systems. Franzoni et al. (2009) described the requirement for fine-grained access control in the Grid environment to support authorizations for flexible job scheduling. Aziz, Arenas, Martinelli, Matteucci, and Mori (2008) detailed the requirement for fine-grained access control in workflow systems in order to provide strong protection at the object level. Minami (2006) illustrated the need for fine-grained access control in order to provide flexible authorizations for mobile users. Steele, Gardner, and Dillon (2007) described the use of fine-grained access control to secure XML data stored in Web accessible repositories. Simpson (2008) discussed the emerging requirements for user-defined, fine-grained access control policies in social networks as a means to address privacy concerns. Cho et al. (2009) described an application of fine-grained access control that facilitates cloaking spatial data. Recently, Wang, Lui, and Wu (2010) and Zhao, Nishide and Sakuri (2011) described the use cryptography to implement fine-grained access control for cloud storage services. Wang, Xiang, Jing, and Zhang (2012) proposed the use of fine-grained access control as an approach to controlling security for Web browser extensions written in JavaScript.

The capability to provide fine-grain access control is now available in a number of popular database products. In the commercial realm, Oracle®, IBM®, and Sybase® offer fine-grained access control as part of their respective database products (Kabra et al., 2006). The Oracle® relational database management system provides fine-grained access control through a transparent query rewrite mechanism known as VPD. Version 9 of the IBM® DB2® relational database product introduced a cell-level fine-grained access control mechanism known as LBAC (Bond et al., 2006). The Sybase EAServer database

system also offers a fine-grained access control capability based upon query rewrite, which is conceptually similar to the Oracle® VPD (Kabra et al., 2006).

Robbert and Ricardo (2003) found that undergraduate database courses typically allocate less than two hours per semester to the topic of database security. Thus, relatively few students are exposed to the theoretical foundations behind advanced security feature such as fine-grained access control. A further complicating factor is that fine-grained access control is a generic approach to providing enhanced database security and encompasses many disparate mechanisms. Significantly, oral tradition in the database community holds that the use of fine-grained access control in relational database management systems imposes significant performance penalties. This viewpoint appears to be supported by recent research. For example, Tolone, Ahn, Pai, and Hong (2005) described the use of role-based security in file-based collaborative systems as providing less overhead than fine-grained security at the user or object level. According to Tolone et al. (1992), the use of fine-grained access control adds complexity, is more difficult to manage, and is highly dependent upon contextual information encompassed within the data to be protected. However, Tolone et al. provided no supporting metrics for this generalization. Zhan et al. (2006) implemented a novel fine-grained access control model for one-variable queries under the PostgreSQL 7.4 relational database management system. Zhan et al. reported 20-30% degradation in query response time when employing a fine-grained access control mechanism based upon query modification. Zhu et al. (2008) evaluated a novel approach to fine-grained access control conceptually similar to the Hippocratic database implementation described by LeFevre et al. (2004). Zhu et al. measured performance degradation in excess of 25% for queries against large tables

using their approach to fine-grained access control termed the “Enforcing Rule”. In contrast, other studies, including LeFevre et al. (2004), Kabra et al. (2006), Zhu and Lü (2007), and Pan (2009), reported implementations of fine-grained access control for database systems with minimal performance impacts.

According to LeFevre et al. (2004), query complexity is an important consideration in quantifying the performance of fine-grained access control. Both LeFevre et al. and Kabra et al. (2006) have identified query optimization (i.e., query simplification) as a factor that can significantly reduce the performance overhead inherent in fine-grained access control. In the study conducted by the author, an estimate of query complexity was employed as one of the independent variables during benchmarking trials.

Barriers and Issues

Rosenthal, Dittrich, Donahue, and Maimone (2001) described the general state of database security research as moribund. However, since 2001, there has been revived interest in database security research, motivated largely by privacy concerns and by the need to secure database systems accessed from the Internet (Zhu & Lü, 2007). Still, the total amount of research in this subject area of database security is somewhat sparse. Some of the most influential research papers on the topic of fine-grained access control, for example Agrawal et al. (2002), LeFevre et al. (2004), and Rizvi et al. (2004), originate from the early years of the previous decade. Nevertheless, the foundational work by these researchers continues to be relevant to current research, as seen for example in the work by Zhang (2008), Zhu et al. (2008), Siegenthaler and Birman (2009),

and Yu (2009).

Agrawal et al. (2002) introduced the concept of the Hippocratic Database. LeFevre et al. (2004) in her seminal paper describing an implementation of the Hippocratic Database technology, focused not only on a pragmatic implementation of the technology, but also on the performance characteristics of Hippocratic Database technology. The work of LeFevre et al. received significant discussion and analysis in subsequent research work on fine-grained access control for database systems by Bertino et al. (2005), Kabra et al. (2006), Wang et al. (2007), Johnson and Grandison (2007), and Zhu et al. (2008).

The seminal paper of Rizvi et al. (2004) was highly instrumental in generating renewed research interest in the area of fine-grained authorizations. Rizvi et al. proposed the Truman and Non-Truman models to describe the behavior of different approaches to fine-grained authorization. The concepts embodied in the Truman model have influenced research in the area of fine-grained access control for database systems, as may be seen in the work of Purevjii, Aritsugi, Imai, and Kanamori (2007), Wang et al. (2007), Olson et al. (2008), Corcoran, Swamy, and Hicks (2009), and Zhang et al. (2009). As well, the concept of the Truman and Non-Truman models has influenced the development of fine-grained authorizations for XML; this is reflected in the work of Kanza, Mendelzon, Miller, and Zhang (2006), Steele et al. (2007), and Currim et al. (2009).

Although there is a considerable body of recent work focused on the issue of fine-grained access control, the number of researchers pursuing topics in this field is relatively small. Thus, much of the literature referenced in the study was drawn from a relatively small group of researchers. Nevertheless, the research available on the topic of fine-

grained access control is both relevant and important. Rosenthal and Winslett (2004) described concerns in the area of database security research as follows "…progress in data security has been slow, and (too) much security enforcement is in application code, or else is coarse grained and insensitive to data contents" (pp. 962).

The study undertaken by the author provides new information concerning the performance and scalability of four common fine-grained access control mechanisms. This information should be of considerable benefits to database professionals and system architects looking to secure relational database management systems accessed from the Internet. In addition, information concerning the scalability of four common fine-grained access control systems should help address possible concerns about the use of this technology for real-world applications. The work undertaken in this study builds upon previous database security benchmarking as reported by LeFevre et al. (2004), Kabra et al. (2006), Zhu et al. (2006), and Zhu and Lü (2007).

Limitations and Delimitations of the Study

The study that was undertaken employed performance benchmarking as a mechanism for quantifying the performance cost imposed by database fine-grained access control. The intent was to measure the relative performance of relational database management systems that implemented fine-grained access control, compared against like configurations that did not employ fine-grained access control. According to Gray (1992), benchmarking is often employed as a methodology to quantify the performance of a computer system using just a single numerical value (e.g., transactions per second).

Metrics generated through the benchmarking process were the sole input used to formulate the generic performance model developed in the author's study.

Given that no generally accepted standard exists for the implementation of fine-grained access control in relational database management systems, selection of the approaches to be evaluated were somewhat subjective. The four approaches selected for evaluation included authorization views, a Hippocratic Database, transparent query rewrite, and LBAC. These are the most commonly utilized approaches to fine-grained access control in database systems as described by Bertino et al. (2006), Wang et al., 2007, and Zhu and Lü (2007).

In the recent literature, a concept termed "correctness criteria" has been identified as an issue of significant importance (Rizvi et al, 2004; Bertino & Sandhu, 2005; Bertino et al, 2006; Wang et al., 2007). Correctness criteria pertain to the completeness of results retuned by database queries that are filtered by fine-grained access control. While the author recognizes the significance of correctness criteria, the generic model does not address this issue given that none of the approaches to fine-grained access control selected for evaluation currently satisfies the correctness criteria. Nor in fact are there any fine-grained access control mechanisms commonly in use today that satisfy the correctness criteria (Wang et al., 2007).

A key question that was addressed in the author's study concerns the issue of scalability. Do the performance characteristics of fine-grained access control implementations change significantly when load increases? Elnikety et al. (2009) demonstrated that it is possible to estimate database scalability using small-scale systems and the TPC-W benchmark. Others, including Vieira and Madeira (2005), Kabra et al.

(2006), and Zhu and Lü (2007) have evaluated TPC benchmarks on small-scale processors to measure the overhead imposed by security mechanisms. According to Menascé (2002), the TPC-W benchmark may also be used to estimate scalability, calculated as the ratio of concurrent sessions versus the cardinality of the ITEM table. The study undertaken utilized the TPC-W benchmark on a small-scale processor to derive metrics for use in formulation of a generic model. Benchmarking on large-scale systems is beyond the scope and resources of the current study.

Definition of Terms

The subject of database security employs a number of terms and phrases that have specific meanings within the field. Definitions for terms and phrases have been drawn primarily from textbooks and research papers. The classic textbook 'Database Security' (Castano, Fugini, Martella, & Samarati, 1994) provides an excellent general reference for many of the specific terms relating to database security. The following definitions are provided to clarify terms used in this document.

Access Control	The process of restricting access to data based upon a predefined access policy (Benantar, 2005).
Authentication	The process of uniquely identifying a system user (Castano et al., 1994)
Authorization	Also termed Access Authorization. Both terms are synonymous with Access Control (Benantar, 2005).
Base Table	A database structure created via the SQL CREATE TABLE statement and stored as an ordered file in

	the Relational Database Management System is termed a base table. (Elmasri & Navathe, 2010).
Benchmark	The process of acquiring measurements for use in comparing performance between systems (Jain, 1991).
Cell	In a relational table or spreadsheet, the intersection between a column and a row (cell, n.d.).
Closed World	An approach to database security that denies access to objects unless an authorization is explicitly specified (Bertino & Sandhu, 2005).
Conditional Validity	The concept that a query can be fully answered using data in a set of database views (Rizvi et al., 2004).
DBA	Acronym for Database Administrator (Castano et al., 1994).
Data Privacy	Concerning the rights of individuals to determine how, when, and under what conditions personal information may be released (Agrawal et al., 2002).
Data Security	Data is said to be secure when the confidentiality, integrity, and availability of the data can be ensured (Bertino & Sandhu, 2005).
Execution Plan	Describes how a SQL query is to be executed including access methods for each database object

	as well as references to the internal algorithms used to implement SQL operators (Elmasri & Navathe, 2010).
Fine-grained	In access control, referring to the granularity of access to data, also known as low-level access under conditions where access to data is restricted at the row, column, or cell level (Wang et al., 2007).
Generic Model	A paradigm used to illustrate concepts at a high level of abstraction using concise definitions (Melnik et al., 2003).
Hippocratic Database	A conceptual database architecture or design that facilitates data privacy (Agrawal et al., 2002).
Index-Scan	Accessing rows in a database table using an index (Garcia-Molina, Ullman, & Widom, 2008).
INGRES®	A prototype relational database management system implemented at the University of California between 1975 and 1977 (Stonebraker & Rowe, 1986).
Join	The SQL JOIN operation combines rows from two tables based upon a column or columns common to both tables (i.e., join-predicates). A JOIN operation returns only the rows satisfying the JOIN condition. (Elmasri & Navathe, 2010).
Label	Commonly used as a synonym for “access class” in

	the MAC model (Bertino & Sandhu, 2005).
Mechanism	An implementation of a model, construct, or paradigm used to provide access control (Benantar, 2005).
Metadata	Descriptive information about the database structure and database contents (Elmasri & Navathe, 2010).
Middleware	A shared application external to the database that provides database connectivity and related services (Elmasri & Navathe, 2010).
Null	A special data value employed to indicate that data contains no specific value (LeFevre et al., 2004).
Performance	A metric quantifying the work performed by a system – e.g., database transactions per second (Gray, 1992).
Object	A passive entity, for example, a table in a relational database management system (Benantar, 2005).
ODBC	An acronym for Open Data Base Connector – an industry standard for interfacing application software to relational database management systems (Elmasri & Navathe, 2010).
Parse Tree	A tree-like structure representing the lexical elements and syntactical categories of a SQL statement (Garcia-Molina et al., 2008).

Policy	A rule used to control access (Castano et al., 1994).
Query Optimization	The assembly of a plan for executing a SQL query efficiently. The SQL query may be simplified as part of the assembly process. The term is a misnomer in that the optimized query is not necessarily optimal, merely adequately efficient (Elmasri & Navathe, 2010).
Query Rewrite	The process where the query parse tree is converted to an algebraic representation of the query (Garcia-Molina et al., 2008). The primary function in query rewrite is to handle views, correctly eliminating duplicate statements, nested queries and NULLs (Hellerstein et al., 2007).
Resource	Encompasses both hardware and software (e.g., CPU, disks, program files). In terms of database access control, authorization for resources encompasses facilitating access to data and database stored procedures (Castano et al., 1994).
Response Time	The elapsed time between submission of the query and receipt of the query response (Elmasri & Navathe, 2010).
Row-level security	An approach to enforcing data access restrictions at the row level (Wang et al., 2007). Using row-level

	enforcement, the entire row is prohibited from viewing even if only a single cell within the row contains confidential data (LeFevre et al., 2004).
Scalable	A solution that is equally applicable to small and large computer systems is said to be scalable (Gray, 1992).
SPEC	An acronym for Standard Performance Evaluation Corporation. A nonprofit consortium of computer vendors that provides computer benchmark programs (Gray, 1992).
SQL	An acronym for Structured Query Language. SQL is the query language used in relational database management systems (Elmasri & Navathe, 2010).
Subject	A term to identify a program in execution that has both and identity and privilege (Benantar, 2005).
Synthetic Workload	A workload with characteristics similar to real world processing that can be executed multiple times in a controlled fashion (Jain, 1991).
System	A general term that encompasses hardware, software and firmware (Jain, 1991).
System R	A prototype relational database management system developed by IBM® during the 1970's (Astrahan et al., 1976).

TCB	Acronym for Trusted Computing Base; defined as the components that enforce the security policy of a system (Bishop, 2002).
Truman Model	Named after the artificial world of Truman Burbank in the movie 'The Truman Show', the model provides "…each user with a personal and restricted view of the complete database" (Rizvi et al., 2004, p. 553).
TPC	Acronym for the Transaction Processing Performance Council. The TPC defines and audits performance and database benchmarks for industry (Transaction Processing Performance Council, 1998).
TPC-W	The TPC Benchmark™ W (TPC-W) is designed to model an on-line bookstore (Transaction Processing Performance Council, 2003).
User	Typically a human being. A user is often associated with a user account or profile which contains both authentication and authorization information for the specific user (Benantar, 2005).
Views	Queries stored in the database catalog, expanded at run-time by the query rewriter, providing dynamic, virtual tables (Hellerstein et al., 2007).

View Replacement	An approach in which the base relations in a query submitted by a user are replaced by authorized views (Kabra et al., 2006).
Wisconsin Benchmark	Developed in 1983 at the University of Wisconsin, a benchmark that is widely used to test the performance of relational database management systems (Gray, 1992).
Workload	A user request made to the system. A user can be either a person or a software program (Jain, 1991).

Summary

The problem that was studied in this dissertation concerns the performance penalties associated with the use of fine-grained access control in relational database management systems. By 1976, fine-grained access control capabilities had been implemented for both the INGRES® and System R prototype relational databases (Griffiths & Wade, 1976). Yet despite the fact that more than 30 years have passed, database fine-grained access is not widely used, although the associated benefits of the technology have been well documented (Rizvi et al., 2004; Agrawal et al., 2007; Roichman & Gudes, 2007; Simpson, 2008; Currim et al., 2009). To date, relatively little known effort has been directed toward the performance issues associated with the use of fine-grained access control (Zhang, 2008). Still, some researchers believe that fine-grained access control solutions for relational database management systems can provide

enhanced security as well as improved privacy protection with only minimal performance penalties (Yu, 2009).

The study undertaken by the author led to the development of a generic model to represent the performance aspects of four approaches to fine-grained access control implemented within a relational database management system. The formulation of a generic model that quantifies the performance penalties associated with the use of fine-grained access control in database systems is intended to provide database administrators and system architects with a much better understanding of the performance implications attendant with the use of these technologies.

Chapter 2

Review of the Literature

Salkind (2006) described the literature review as a chronological investigation of the development of ideas in the subject field, providing an opportunity to examine ideas that were proven true and those that "…were left by the wayside because of lack of support…" (p.43). Boote and Beile (2005) described the literature review as providing the researcher with an understanding of the strengths and weaknesses of previous research, and an opportunity for critical analysis of past work. Sekaran (2003) described the objective of the literature review as the formulation of a research based foundation for the theoretical framework of the proposed study.

The review of the literature is separated into six sections: the contribution of INGRES® and System R, database access control concepts, relational query processing, fine-grained access control, performance benchmarking, and performance models. In the first section of the literature review, the author examines a number of the architectural decisions made in the INGRES® and System R prototype databases that continue to have significant impact on modern relational database management systems. In the second section, the author reviews the technical concepts and paradigms that underlay database access control, focusing on concepts central to the implementation of fine-grained access control. In the third section, the author describes the processes behind relational query

processing, focusing in particular on the function of the database query rewriter, which is a critical component in determining the efficiency of fine-grained access control. In the fourth section of the literature review, the author reviews the current state of fine-grained access control for relational database management systems. In the fifth section of the literature review, the author examines the use of performance benchmarking for quantifying the efficiency of fine-grained access control implementations. In the sixth and final section of the literature review, the author examines the use of performance models as tools for quantifying database performance and scalability.

The Contribution of INGRES® and System R

"Descendants of the early relational prototypes [INGRES® and System R] have become the primary commercial relational DBMSs" (Stonebraker, 2008, p. 76). This is not to imply that there have been no technical advancements in relational database technology since the mid-1970s. Rather, three decades of database research and commercial software development have advanced relational database technology to the point where relational database management systems are now ubiquitous and often considered mission-critical (Hellerstein et al., 2007). Still, it is worthy to note that market forces have provided the primary motivation for database research over the past three decades (Stonebraker, 2008). While advances in relational database technology have provided greatly enhanced reliability, scalability and many new database features, relatively few changes have been made to the underlying architecture of relational database management systems since the INGRES® and System R prototype databases (Hellerstein et al.).

Stonebraker and Wong (1974) described the capability to deploy fine-grained access control in the INGRES® prototype database using query rewrite. According to Stonebraker and Wong, the objectives in defining the access control system for the INGRES® database were that such a system was to be powerful, flexible and impose minimal overhead in terms of processing requirements. Query rewriting is performed at the query language level, allowing queries to be filtered in order to exclude any rows the query is not authorized to return (Bertino et al., 2005). This approach to fine-grained access control is also known as row-level security or content-based access control (Bertino & Sandhu, 2005). Stonebraker and Wong leveraged the database query rewriter, an internal database subsystem used primarily for view compilation, in order to modify query statements without the knowledge of the user, thereby providing effective row-level security. However, this approach to transparent query rewrite suffers from a significant problem in that the results returned from a database query may contain fewer rows than the user is actually authorized to view (Rizvi et al., 2004). Even so, transparent query rewrite continues to be used today. Oracle® VPD, an access control mechanism available in the enterprise version of the Oracle® relational database management system, employs transparent query rewrite in order to provide fine-grained access control (Currim et al., 2009).

Query processing is another area where modern database systems retain a strong link to the System R prototype database. According to Deshpande, Ives, and Raman (2007), cost-based query optimization developed for System R continues to be used extensively in modern relational database management systems. Selinger, Astrahan, Chamberlin, Lorie, and Price (1979) described the query optimizer in System R as

incorporating the use of three categories of metadata for optimizing queries: cardinality estimates, execution plan cost estimates, and searches for optimal execution plans. According to Chaudhuri (2009), in spite of 30 years of continuous improvements, cost-based query optimization continues to be based upon the System R "…query in, plan out model" (p. 967). However, Hellerstein et al. (2007) states that the work by Selinger et al. (1979) was preliminary research and therefore, most modern relational database management systems contain some optimizations in the areas of selectivity, search algorithms, and parallelism to address known limitations in the System R query optimizer. Further, Deshpande et al. cautioned that the System R style of query optimization could break down when provided with insufficient statistical information resulting in generation of optimizer errors at a rate exponential to the size of the query.

View replacement using authorization views is a technique frequently employed to implement fine-grained access control in relational database management systems. Kabra et al. (2006) described view replacement as an approach where user queries are redirected to authorization views rather than being permitted as queries against base tables. According to Kabra et al., fine-grained access control that employs a view replacement model depends heavily on query optimization for good performance.

Views in System R were introduced primarily as a means of providing read-only authorizations on tables; however, System R also allowed updates on views referencing a single underlying base table (Astrahan et al., 1976). Conceptually, a view in System R was a stored query, providing access to a subset of rows and columns on one table or series of joined tables (Blasgen et al., 1981). System R also introduced parameter driven authorization views, thereby providing a filtering mechanism for content-based

authorizations (Bertino & Sandhu, 2005). Rizvi et al. (2004) described conventional authorization views, including authorization views incorporating dynamic parameter selection, as the "Truman Model" of fine-grained access control. Under the Truman Model, query results returned to the user may be incomplete. This condition is resolved in an alternate query model that Rizvi et al. described as the "Non-Truman Model".

Views were stored in the System R database in a format known as a pre-optimized package (Astrahan et al., 1976). The System R pre-optimized package was comprised of a SQL parse tree and a query execution plan (Astrahan et al.). For a query against a view in System R, the user-supplied query was merged with the stored parse tree using query rewrite (Blasgen et al., 1981). According to Bertino and Sandhu (2005), this approach, termed "view composition", continues to be used in modern database systems. However, a side effect of view composition concerns inefficiencies that may be introduced when the SQL WHERE clause of the stored view and the SQL WHERE clause of the user query are combined (Bertino & Sandhu). Halevy (2001) indicated that the view composition approach works best when base tables referenced in the query fully intersect with a view referencing the same base tables. Otherwise, additional query rewriting may be required to eliminate redundant joins, thus resulting in increased query compilation and lengthy execution times (Kabra et al., 2006).

The original user authorization subsystem in the System R database required that access to tables and views to be explicitly granted or revoked by the object owner or by a database administrator (Blasgen et al., 1981). Griffiths and Wade (1976) proposed an enhanced scheme of SQL based GRANT and REVOKE operations for tables and views in the System R database. The scheme proposed by Griffiths and Wade allowed access

privileges to be delegated to other users in order to streamline security management. This scheme of coarse-grained (e.g., table level authorizations) eventually evolved into what is now termed role-based access control (RBAC). According to Bertino and Sandhu (2005), by the mid-1990s, RBAC had become the standard for coarse-grained security in most relational database management systems. The use of roles provides flexible management of user authorizations. Under RBAC, object permissions are assigned to roles, and roles are assigned to users, thereby negating the onerous requirement that object permissions be assigned to individual database users (Bertino & Sandhu, 2005).

Denning, Akl, Heckman, Lunt, Morgenstern, and Neumann (1987) described a hierarchical view implementation for the System R database allowing classification of data and the enforcement of mandatory security. The approach described by Denning et al. (1987) employed database views as objects to which authorizations were applied. The hierarchical or multilevel approach described by Denning et al. for System R employed three sets of views:

1. A base set of views classified data by security level – top-secret, secret, confidential, and unclassified;
2. An intermediate sets of views served to classify aggregations of data that had a higher security classification than the constituent elements;
3. A tertiary set of views that removed (filtered) data classified at a higher level than the access level of the user querying the data.

The INGRES® and System R databases were highly influential in establishing relational databases as the primary database technology of today (Garcia-Molina et al., 2008). According to Bertino and Sandhu (2005), the System R discretionary access

control model and its extensions provide the basis for coarse-grained access control in modern relational database management systems. As well, the work by Denning et al. (1987) using the System R prototype database demonstrated a successful proof of concept deployment of mandatory access control in a relational database management system. Zhang (2008) noted that today, all major database vendors, including Oracle®, IBM®, and Microsoft®, currently provide mandatory access control mechanisms for their respective database products using LBAC.

Access Control Concepts

The internal protection of database systems is provided by the combination of three subsystems – authentication, authorization, and auditing (Castano et al., 1994). Figure 1 illustrates the relationship of these three subsystems. The terms authorization and access control are considered synonymous, although the term authorization is much more widely used (Benantar, 2005). However, the term access control may also be employed in a broader context to describe the protection state of a system (Bishop, 2002). According to Bishop, "protection" describes the conditions "…under which a system is secure" (p. 31). Authentication is the first internal subsystem involved in protecting the database. In order to use services provided by a relational database management system, the user must first be authenticated (Castano et al., 1994). Bishop (2002) describes authentication as the process where an external entity, the user, must confirm their identity. In most cases, the term user is associated with a human being (Benantar, 2005).

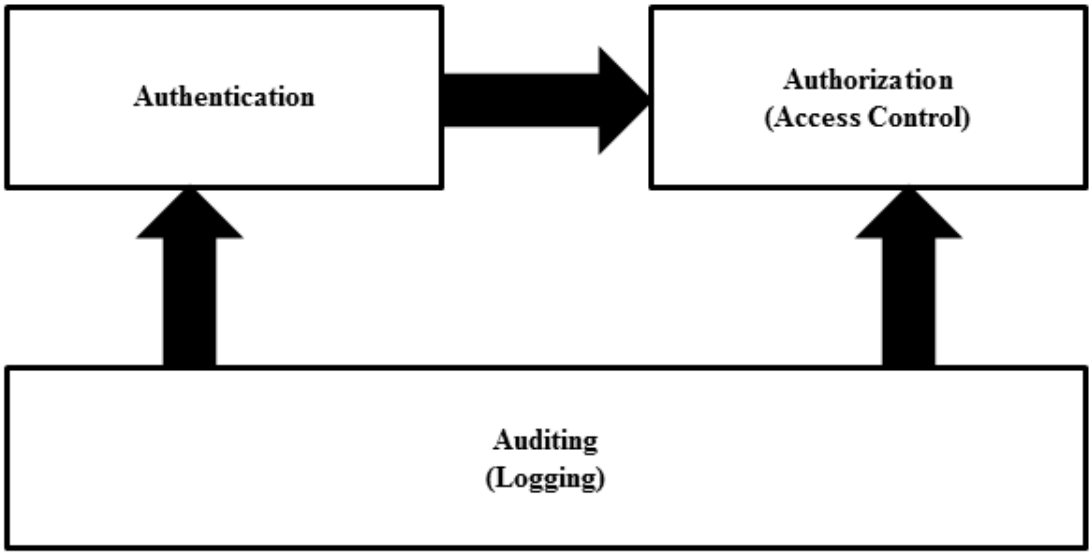

Figure 1. Internal Protection of Database Systems (adapted from Benantar, 2005, p. 17)

Authorization is the second internal subsystem involved in protecting the database. However, authorization cannot occur until authentication is successfully completed. According to Benantar (2005), authorization (also termed access control) is the process of granting user access to data based upon a security policy. By default, users should have no access to data until suitably authorized. This paradigm, where security policies only contain access condition but no prohibitions, is termed a closed world system or closed world model (Castano et al., 1994). Standard coarse-grained security in modern relational database management systems is implemented under a closed world model (Castano et al.). In contrast, fine-grained access-control supports the open world model, in which both access rights and access prohibitions may be explicitly specified in security policies (Bertino & Sandhu, 2005). It is worth noting however, that closed world systems are typically considered more secure than open world systems as the authorization state is more easily verified (Castano et al.).

Auditing is the third internal subsystem used to protect the database. Audit logs are useful in identifying unauthorized transactions if database tampering is suspected (Elmasri & Navathe, 2010). Audit logs can be used in database systems to record both read and write operations, although there are practical constraints on the granularity of event logging imposed by log storage considerations (Castano et al., 1994). According to Bishop (2002), auditing is often considered external to the database protection model as auditing provides accountability, but does not directly protect data.

Access control models can be classified into three broad categories: discretionary access control (DAC) models, mandatory access control (MAC) models, and role-based access control (RBAC) models (Bishop, 2002). DAC models provide an identity-based approach to security where the object owner has complete discretion over who may access an object (Benantar, 2005). However, DAC also allows the object owner to delegate access control responsibility on an object to other database users. MAC models classify both data and users based on security classes (Elmasri & Navathe, 2010). MAC models restrict access of subjects to objects based on the use of hierarchical labels (Castano et al., 1994). RBAC models are employed to restrict access based upon job function (Bishop, 2002).

Under DAC, an authorized user may grant or revoke access rights on objects to other system users (Elmasri & Navathe, 2010). DAC for relational database management systems also embodies the concept of delegated administration using the SQL GRANT option (Bertino & Sandhu, 2005). The flexibility inherent in the DAC model provides a de facto mechanism for decentralized administration of authorizations (Castano et al,

1994). However, this same flexibility is also an inherent weakness in the DAC model in that there are no controls on how access rights are propagated (Bertino & Sandhu, 2005).

According to Castano et al. (1994), the underlying theoretical foundation of the DAC model is the access matrix model. Elmasri and Navathe (2010) describe the access matrix model as consisting of columns, which represent objects (e.g., base tables) and rows, which represent subjects (e.g., users). The intersection of a row and a column in the access matrix model, known as a cell, denotes the privileges of a subject with respect to an object (Bishop, 2002). The structure of the access matrix model can be logically extended to secure objects (e.g., tables and views) in a relational database (Castano et al.). Zhang (2008) provided the following description of the access matrix model applied to database security: "Each cell in the matrix has a flag indicating whether the user at that cell is able to read or update the corresponding data item" (p. 8).

In contrast to the DAC model, the MAC model uses a mechanism termed the trusted computing base (TCB) for enforcement of access control (Benantar, 2005). The TCB encompasses all of the mechanisms, including hardware, software, and firmware, used to enforce security policies (Bishop, 2002). "Neither the subject, nor the owner of the object can determine whether access is granted" (Bishop, p. 104).

The Bell-LaPadula model provides the theoretical foundation for the MAC model (Bertino & Sandhu, 2005). According to Castano et al., the combination of the simple security property and the star property in the Bell-LaPadula provides restrictions to ensure that data will never flow from a higher security level to a lower security level. MAC requires that objects be protected from unauthorized access as well as from

disclosure through unintended means such as flow violation and inference (Castano et al, 1994).

Under the MAC model, "…policies regulate access to data by subjects on the basis of predefined classification of subject and objects in the system" (Bertino & Sandhu, 2005, p. 9). MAC employs access classes, more commonly termed labels, as a mechanism for restricting access to data (Castano et al., 1994). The security label associated with the object is compared with the access class of the user in order to determine whether access to the object is permitted (Benantar, 2005). According to Bertino and Sandhu, fine-grained access control is supported in the MAC model with row-level labels. LBAC in Oracle® and IBM® conforms to the MAC model.

Under RBAC, privileges are assigned to roles rather than to individuals (Elmasri & Navathe, 2010). A role is intended to be an abstract representation of a set of responsibilities, typically representing a job function (Bishop, 2002). Given that security policies are often formulated based on responsibilities or job functions, RBAC can be employed to generalize many real-world security requirements (Benantar, 2005).

RBAC is the coarse-grained access control mechanism that is the default security provided by most modern relational database management systems (Elmasri & Navathe, 2010). The strength of RBAC is that it can be employed to represent the hierarchical security requirement of real-world organizations (Bertino & Sandhu, 2005). However, RBAC can also support complex security implementations. According to Elmasri and Navathe, role-based access control can be used to represent both DAC and MAC models.

Relational Query Processing

In order to be feasible, fine-grained access control requires efficient query processing, and in particular, requires efficient query optimization (Zhang, 2008). In Figure 2 the high-level architecture of a typical relational database management system is depicted, including the component sub-systems of the relational query processor. According to Garcia-Molina et al. (2008), the three major steps or phases of relational query processing include:

1. Parsing – creating a parse tree representation of the SQL query;
2. Query Rewrite – transforming the parse tree into a logical query plan;
3. Plan Generation – converting the logical plan into a physical plan.

In the parsing phase, the query is scanned, parsed, and validated. Elmasri and Navathe (2010) described scanning as the identification of SQL keywords, parsing as the process of checking the syntax of the SQL query, and validation as the process of confirming access to database objects referenced in the SQL query. Once the parse tree has been created, the query is optimized. According to Garcia-Molina et al., the query rewrite and physical plan generation phases of SQL query compilation are collectively known as query optimization. The process of query optimization is the most crucial and time-consuming component of query processing in a relational database management system (Chaudhuri, 2009).

Deshpande et al. (2007) stated that effective query optimization depends upon the presence of a stable execution environment and sufficient statistical information collected

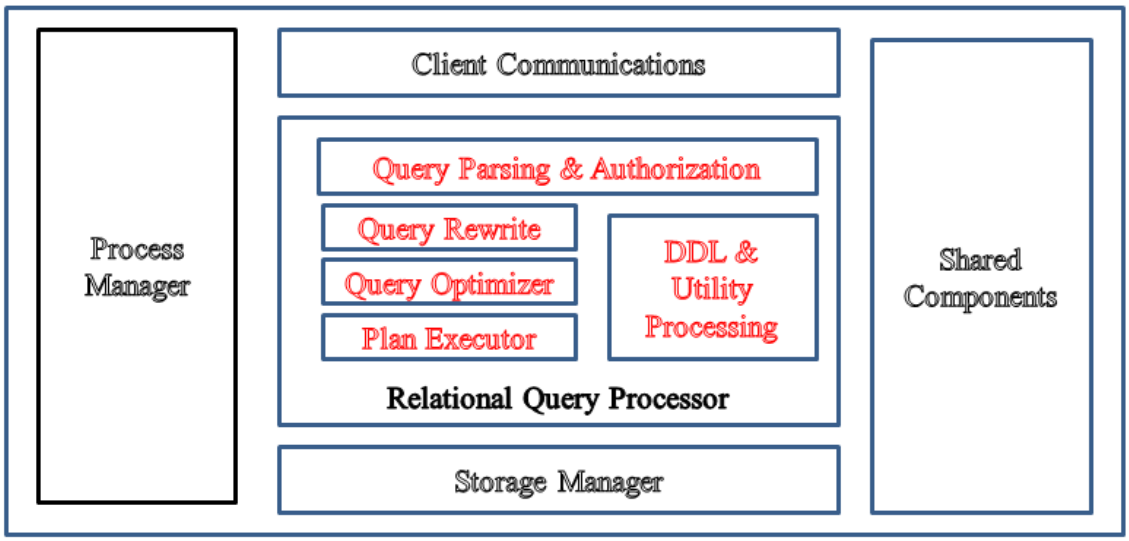

Figure 2. Relational Database Architecture (adapted from Hellerstein et al., 2007, p. 14

for base tables and indexes. During the optimization process, only data in the database catalog (i.e., metadata) is available to the query optimizer (Elmasri & Navathe, 2010). This metadata includes table size (cardinality), related indexes, data location on disk, and frequency of table attributes (Garcia-Molina et al., 2008).

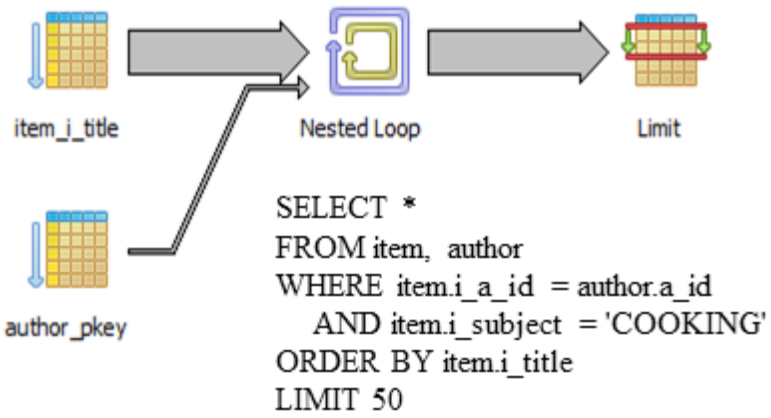

Figure 3. Example TPC-W Query Execution Plan

A query execution plan generated by the database optimizer provides a data-flow like representation of the database query as depicted in Figure 3. In Figure 3, the TPC-W "Subject-Search" SQL query is decomposed showing the two constitute tables, ITEM and AUTHOR. The "Nested Loop" symbol depicted in Figure 3 indicates that a join operation between ITEM and AUTHOR is effected with the ITEM table forming the outer loop and the AUTHOR table the inner loop (Elmasri & Navathe, 2010). Figure 3 also indicates that the join of the AUTHOR table to ITEM utilizes the table primary key (i.e., author_pkkey). The query is limited to the first 50 rows returned as indicated by the "Limit" symbol in Figure 3. Although an ORDER BY clause is included in the SQL query, a separate sort operation is not required in this particular instance as the ITEM table is read in descending primary key (i.e., sorted) order.

According to Hellerstein et al. (2007), each query block in the query execution plan is optimized separately. The calculated cost of each block in the query execution plan is used to determine the cost of the complete query (Hellerstein et al.). According to Elmasri and Navathe (2010), the efficiency of the execution strategy selected is based on the execution plan cost, which is an estimated value, not a measured value. Elmasri and Navathe noted that the query optimization strategy selected by the query optimizer is not necessarily the optimum execution strategy, merely one that is reasonably efficient, albeit suboptimal. However, Chaudhuri (2009) described the current generation of relational database query optimizers as producing a surprising large number of optimal query plans. Reddy and Harista (2005) noted that query plan efficiency (i.e., which plans are deemed optimal) is decided primarily based upon the estimated query response time. On the other hand, Chaudhuri noted that the use of parameterized queries, such as those employed in

some authorization views, could result in the constant regeneration of query execution plans due to changes in the underlying data. As well, Helerstein et al. (2007) noted that the run-time characteristics of parameterized queries could vary significantly depending upon whether parameter values at run-time are typical of the parameter values employed during the initial compilation of the parameterized authorization view.

In addition to graphical query execution plans, a cost-based analysis can also be generated for individual queries in text format as depicted in Figure 4 for the TPC-W subject-search query. Kabra et al. (2006) employed query cost analysis to estimate

```
Limit  (cost=0.00..3966.91 rows=50 width=972)
       (actual time=1.273..39.685 rows=50 loops=1)
 Output: item.i_id, ...
 Buffers: shared hit=101 read=1262
 -> Nested Loop  (cost=0.00..32599546.12 rows=410893 width=972)
                 (actual time=1.268..39.582 rows=50 loops=1)
    Output: item.i_id, ...
    Buffers: shared hit=101 read=1262
    -> Index Scan using item_i_title on public.item
         (cost=0.00..30002110.87 rows=410893 width=574)
         (actual time=1.140..34.910 rows=50 loops=1)
       Output: item.i_id, ...
       Filter: ((item.i_subject)::text = 'COOKING'::text)
       Buffers: shared hit=2 read=1161
    -> Index Scan using author_pkey on public.author
         (cost=0.00..6.31 rows=1 width=398)
         (actual time=0.078..0.080 rows=1 loops=50)
       Output: author.a_id, ...
       Index Cond: (author.a_id = item.i_a_id)
       Buffers: shared hit=99 read=101
Total runtime: 39.954 ms
```

Figure 4. Cost analysis of TPC-W Query

query complexity for implementations of fine-grained access control. Both Kabra et al. and LeFevre et al. (2004) manually simplified SQL queries to reduce complexity (i.e.,

reduce the cost of query execution) as part of implementing efficient fine-grained access control mechanisms in their respective fine-grained access control implementations for SQL Server® and DB2®. However, it was explicitly understood by both authors that the database query optimizer would further revise their simplified SQL statements prior to execution.

Database Fine-Grained Access Control

For maximum effectiveness, fine-grained access control should be an integral part of the database software, rather than custom code developed as part of an individual application (Rizvi et al., 2004). Providing fine-grained access control at the database level ensures that all access to the database is subject to the same access control regime (Wang et al., 2007). As well, fine-grained access control implemented within a relational database management system is far more difficult to subvert than fine-grained access control implemented externally through custom application code (Zhu & Lü, 2007). Further, if fine-grained control exists only at the application level or middleware level, the entire database may be subject to compromise (Kabra et al., 2006).

Enforcement of fine-grained access control involves two components – a mechanism and a security policy (Siegenthaler & Birman, 2009). A mechanism is a process to enforce security policies (He & Yang, 2009). In more specific terms, "An access control mechanism refers to a particular method, tool, or procedure for implementing an access control policy" (Benantar, 2005, p. 26). In the context of access control, a security policy comprises a set of rules (Benantar, 2005). Using the information specified in a security policy, an access control mechanism either grants or denies user

access to data (Kocaturk & Gundem, 2008). However, the security policy must be sufficiently fine-grained to express all access control requirements (Fischer, Marino, Majumdar, & Millstein, 2009). On the other hand, Bell (2005) cautions that the granularity of a security policy can be no greater than the granularity supported by the algorithms underlying the access control mechanism.

Zhang (2008) noted that the use of fine-grained access control in relational database management systems imposes performance issues. According to Kabra et al. (2006), most of the existing approaches to fine-grained access control, including existing commercial implementations, effectively replace a query on a base table with a query on a view of the base table in order to remove unauthorized data. This category of mechanism is termed a view-based or view replacement approach. According to Zhang, the view-based approach imposes additional overhead in query processing due to the requirement to rewrite user queries to enforce fine-grained access control.

Zhang (2008) identified three areas in query compilation and query processing where the use of fine-grained access control imposes additional overhead:

1. Query rewriting to enforce fine-grained access control changes the query semantic structure, producing a query plan that is potentially less optimal than the original;
2. The view-based approach to removing unauthorized data imposes additional complexity on the user query. With the added complexity, the query optimizer needs to determine the cardinality of both the original query and the rewritten query with fine-grained access controls enforced.

3. At run-time, additional authorization checks are required due to the user query being rewritten to enforce fine-grained access control.

Figure 5 depicts the query evaluation steps that take place when fine-grained access control is absent and when fine-grained access control is implemented. According to Zhang, cardinality is one of the most important factors affecting query performance when fine-grained access control is implemented. Zhang, Ilyas, and Salem (2009) describe Partitioned Sampling for Multiple users (PSALM), a technique that improves query

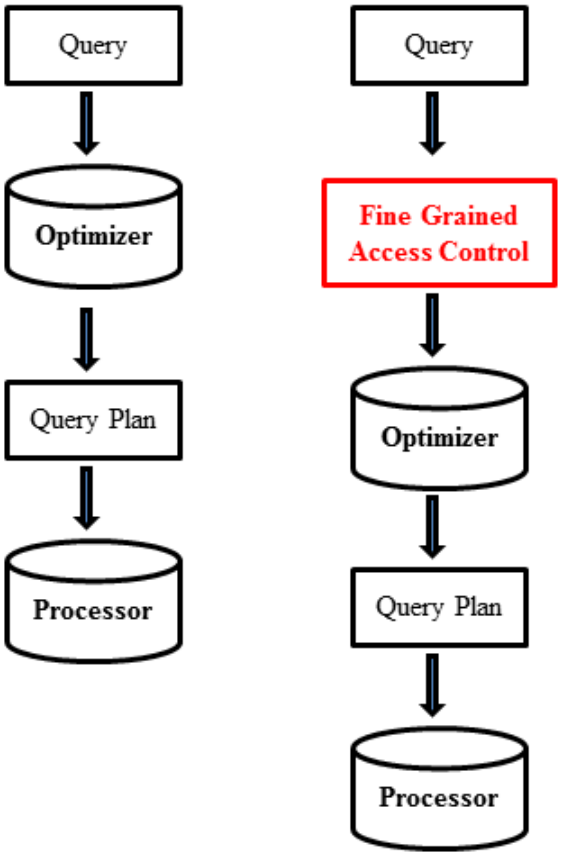

Figure 5. Query Evaluation in Relational Databases (adapted from Zhang, 2008, p. 3)

performance under systems that implement fine-grained access control. Zhang et al. (2009) demonstrated that query efficiency under fine-grained access control could be

improved if the cardinality of accessible rows can be estimated with a high degree of accuracy at the time the query is compiled.

Parameter driven authorization views conforming to the Truman model are useful in constructing dynamic views that can be embedded in Web applications (Rizvi et al., 2004; Roichman & Gudes, 2007). However, authorization views may incur additional overhead at run-time as dynamic parameters provided through user input may contain values substantially different than the estimated values stored in the database at the time the authorization view was originally compiled (Hellerstein et al., 2007). As well, separate authorization views may be required for SQL SELECT, INSERT, UPDATE and DELETE operations (Bertino & Sandhu, 2005; Kabra et al., 2006). According to Bertino and Sandhu, providing sufficient fine-grained security within an application may require multiple versions of authorization views to be maintained, which thus limits the practicality of authorization views as a general approach to fine-grained access control.

A common approach to enforcing privacy protection at the cell level involves replacing restricted attributes with a "null" value (Bertino et al., 2005). Null is a special value in standard SQL, generally assumed to indicate that the attribute has no defined value (Garcia-Molina et al., 2008). However, according to LeFevre et al. (2004), null may also be employed to indicate absence of an attribute; for example, a null might appear in a column containing phone numbers to indicate that no telephone was installed. Further, null cannot be used as a value in a primary key, an issue that may limit the applicability of nullification as a data masking technique (LeFevre et al.). According to Zhu et al., (2008), in nullifying cells containing sensitive data, the act of nullification

itself indicates that sensitive information exists in the original data, providing possible incentive for further attacks.

Garcia-Molina et al. (2008) identified a number of considerations associated with the use of nulls in ANSI Standard SQL. According to Garcia-Molina et al., these considerations include:

1. SQL operations using outer joins may return nulls where no values exist;
2. Comparing a null value to any other value, including another null value, evaluates to unknown;
3. The proper method to determine whether a cell contains a null value is to employ the IS NULL operator, which returns a Boolean TRUE or FALSE.

Kabra et al. (2006) discussed the performance issues associated with the use of nullification when used for privacy enforcement. The key issue identified by Kabra et al. concerns the impact of query rewrite on the original query and the likelihood that redundant authorization checks will be appended to the rewritten query. However LeFevre et al. (2004) found that while outer joins used to implement nullification for privacy protection can be inefficient, the judicious use of secondary indexing could negate some performance concerns. In addition, LeFevre et al. found that for large I/O bound queries, for example queries executed against large tables, the privacy-checking component of the query is a CPU intensive process, and thus can be addressed effectively through processor scaling. Significantly, the process of query simplification, as for example in the tuning of authorization views, can sometimes result in better performance than provided by the query plan of the original query (LeFevre et al.; Kabra et al.).

Rizvi et al. (2004) introduced the concept of unconditional validity when dealing with authorization views. According to Rizvi et al., unconditional validity requires that data returned from an authorization view be identical to data returned from the same query executed directly on the base table or tables. Rizvi et al. proposed the Truman and Non-Truman models of fine-grained access control based upon validity of query results returned. According to Bertino et al. (2005), the Truman model may not always return complete results and is therefore conditionally valid, whereas the Non-Truman model is unconditionally valid as it always returns complete query results.

Bertino et al. (2005) proposed that an algorithm used to enforce fine-grained access control should be sound, secure, and maximum. Wang et al. (2007) termed these three properties "correctness criteria" (p. 555). According to Bertino et al., fine-grained access control should always return information that is accurate (sound), should not contain unauthorized information (secure), and should return all authorized rows (maximum). Wang et al. argued that most query modification algorithms used to implement fine-grained access control fail to meet the correctness criteria.

The topic of correctness criteria is an active area of research relating to the use of fine-grained access control. Zhu et al. (2008) demonstrated that inclusion of the SQL operators NOT IN and NOT EXISTS in a sub-query can cause additional (i.e., incorrect) information to be returned by the sub-query, resulting in information leakage, thus violating the soundness property. Shi et al. (2009) examined SQL queries used in fine-grained access control and provided a generalization for the soundness property using a relational algebra based analysis technique. Rastogi, Suciu, and Welbourne (2008) examined the need for fine-grained access control in radio frequency identification

(RFID) data management, stating that in order to address the soundness property, fine-grained support for both access and deny security policies is required.

Authorization Views

To enforce restricted access to data, authorization views are the most commonly employed alternative to allowing unrestricted access to base tables in the database. Authorization views can be constructed using conventional SQL view syntax and may include dynamic parameters that are resolved when the view is executed (Rizvi et al., 2004). Using authorization views, user queries are written against a view rather than against the base table, so that the user query only returns data the user is authorized to view (Kabra et al., 2006). As noted by Kabra et al., authorization views may also include a mechanism for authorization checks, which can potentially introduce inefficiencies, thereby resulting in increased query optimization overhead and slower query execution. However, the most significant source of inefficiency when using authorization views relates to the process of view instantiation. Hellerstein et al. (2007) noted that compiled views are stored in the database catalog along with a query execution plan generated at the time of view compilation. However, the data underlying the view may change significantly over time, resulting in poor query performance. Significant data changes affecting query execution could include changes in table cardinality as well as changes in attribute frequency (Garcia-Molina et al., 2008).

Another source of inefficiencies relates to the structure of the SQL statement in authorization views. Hellerstein et al. (2007) warned that although authorizations on views are verified at the time of view compilation, they must also be checked again at the time of query execution, adding additional overhead. According to Kabra et al. (2006),

even though authorization views are pre-compiled, they can incur significant performance penalties at execution time.

Kabra et al. (2006) also noted that authorization checks incorporated in authorization views might be re-written by the query optimizer to include the use of SQL semi-joins. A SQL semi-join between two table is constructed using either the EXISTS or IN comparison operator and a nested sub-query. A SQL semi-join can be quite efficient if the nested sub-query returns only a small number of rows (Elmasri & Navathe, 2010). However, Dietrich (2001) warned that SQL queries incorporating nested sub-queries generally incur more overhead than SQL queries constructed without nesting.

Bertino et al. (2005) commented that the use of authorization views for fine-grained access control is potentially a naïve solution in that a great many authorization views may be required to implement the desired level of granularity for access control. However, Rizvi et al. (2004) proposed the use of parameterized authorization views as a means of reducing the number of authorization views required to provide fine-grained access control for an application. An example of the SQL code for a parameter driven authorization view is depicted in Figure 6. The $user-id run-time parameter shown in the SQL statement in Figure 6 always contains the user-id of the user executing the database query (Astrahan et al., 1976). Roichman and Gudes (2007) proposed the use of parameter driven authorization views to provide fine-grained access control for Web applications.

```
CREATE VIEW studentgrades AS
SELECT grades.*
FROM   grades, registered
WHERE registered.student-id = $user-id
   AND grades.course-id = registered.course-id;
```

Figure 6. Parameterized View under the Truman Model (adapted from Rizvi et al., 2004)

Transparent Query Rewriting

Stonebraker and Wong (1974) were the first to describe the use of transparent query rewrite for fine-grained access control based upon the following principles:

1. The user interacts with the data through a high-level query language;
2. The database engine transparently modifies the query without the knowledge of the user in order to eliminate unauthorized data;
3. The modified query statements are simplified for execution by the database.

Efficient query rewrite depends upon effective query optimization. According to Hellerstein et al. (2007), query optimization is among the most complex functions performed in modern day commercial databases. In modern relational database management systems, both Oracle® and Sybase® employ transparent query rewrite to provide fine-grained access control (Kabra et al., 2006). Oracle® VPD is the most frequently cited example of fine-grained access control using transparent query rewrite (Zhang, 2008).

Under Oracle® VPD, when a user query is issued, the SQL WHERE clause of the query is modified through the addition of predicates. According to Shi et al. (2009), predicates selected are based upon security policies defined as part of the VPD configuration. Currim et al. (2009) described this combination of policy and enforcement mechanism as providing "…a query that is in effect evaluated on a user's private view of the data" (p. 40). Hence Oracle® describes this approach to fine-grained access control as a virtual private database (VPD).

According to Oracle Corporation (2010), "To implement Oracle® Virtual Private Database, you must create a function to generate the dynamic WHERE clause, and a

policy to attach this function to the objects that you want to protect" (p.7-4). Although only available in the Enterprise Edition of the Oracle® relational database management system, VPD can be used to protect tables, views, and synonyms. Oracle Corporation describes the operation of VPD as follows:

1. When the user connects to a VPD enabled database, a log-on trigger fires to load a security policy into the user memory space – this is the VPD security context, which can be examined through a database view;
2. The security policy (security context) can be applied to SQL SELECT, INSERT, UPDATE, and DELETE operations on both tables and views;
3. When the user issues a SQL command to access an object protected by VPD, a predicate is appended to the WHERE clause of the SQL statement;
4. The predicate in the WHERE clause is tested against each row in the table before access is granted.

Wang et al. (2007) noted that in Oracle® VPD, sensitive attribute values returned in the query are masked with NULL values as opposed to simply excluding the entire row as would be the case in a conventional row-level security deployment. According to Zhang et al. (2008), a query under Oracle® VPD is "…rewritten and answered under the Truman model" (p. 19). Bertino et al. (2005) indicated that the Oracle® VPD is a more scalable technology than authorization views due to the dynamic nature of the VPD technology. Although Oracle® VPD is primarily considered as a row-level security implementation, Bertino et al. (2005) indicated that VPD could also be employed for fine-grained column level security.

Hippocratic Databases

Agrawal et al. (2002) introduced the concept of the Hippocratic Database. Just as the Hippocratic Oath directs physicians to preserve the privacy of patient information, privacy-preserving policies stored in the Hippocratic Database provide rules that are used to control access to personal information. The need for privacy preservation provided by Hippocratic Databases is motivated by the emergence of new and increasingly more stringent privacy legislation in the European Union, the United States, Canada, and Australia (Johnson & Grandison, 2007).

The Hippocratic Database proposed by Agrawal et al. (2002) is primarily a conceptual approach to improving database security. Hippocratic Databases are configured using conventional relational database management systems with the addition of a SQL-based query modification mechanism that enforces security policies stored in database tables. In the Hippocratic Database, it is the implementation of the security policy store and relational joins to policy tables that are of the most concern in terms of quantifying query performance costs (LeFevre et al., 2004). According to Agrawal et al., the conceptual elements of the Hippocratic Database may also include such components as a policy editor, a data retention manager, a data collection analyzer, and a query intrusion detector. The provision of auditing to confirm privacy enforcement is also a feature of the Hippocratic Database (Agrawal et al., 2007).

LeFevre et al. (2004) evaluated the performance of two implementations of a Hippocratic Database using the Wisconsin benchmark. In the work reported by LeFevre et al., the effect on query performance was measured for five variables: privacy enforcement mode, database size, query filtering, enforcement model, and query

structure. Experiments were performed using small-scale server hardware and the DB2® relational database management system. According to LeFevre et al., acceptable but statistically significant performance degradation was apparent when privacy protection was enforced using a Hippocratic Database.

LeFevre et al. (2004) evaluated two separate approaches to SQL coding for implementing Hippocratic Database systems. In the first approach, a SQL CASE

```
SELECT
  CASE WHEN EXISTS
    (SELECT  phone.choice
     FROM    patientchoice
     WHERE   patients.pnumber  = patientchoice.pnumber
       AND   patientchoice.phonechoice  = 1)
  THEN phone ELSE null  END
FROM patients  …
```

Figure 7. Query using CASE Statement (adapted from LeFevre et al., 2004, p. 113)

statement containing a sub-select was used to filter results returned. In the example provided in Figure 7, if the value of the privacy flag is one, then the stored value for phone number is returned. If the value of the privacy flag is zero, then a NULL (i.e., no defined value) is returned, thereby masking the true value for the phone number. In the second approach described by LeFevre et al., and as depicted in Figure 8, a sub-select and

```
SELECT  phone
FROM (SELECT  pnumber
          FROM    patientchoices
          WHERE  patients.pnumber  = patientchoice.pnumber
             AND  patientchoice.phonechoice  = 1 )  t1
LEFT  OUTER  JOIN patients  t2 ON t1.pnumber  = t2.pnumber
```

Figure 8. Query using Outer JOIN (adapted from LeFevre et al., 2004, p. 114)

an outer join to a second data table are employed rather than an inline CASE statement. Based upon the value of the privacy flag, the LEFT OUTER JOIN returns either the stored value for the phone number or a NULL value.

In addition, LeFevre et al. evaluated two separate configurations for storing privacy enforcement metadata. In one configuration, the privacy metadata was stored in base tables by appending privacy metadata columns to the base table. In the second configuration described by LeFevre et al., privacy metadata was stored in separate tables used only for metadata. According to LeFevre et al., use of a SQL CASE statement, combined with storage of privacy metadata in base tables, generally provides the best performance – however, to satisfy complex privacy requirements, queries employing complex joins to external metadata tables may be more efficient.

Johnson and Grandison (2007) described an implementation of the Hippocratic Database technology using a middleware component dubbed “Active Enforcement”. The Active Enforcement middleware is not tied to a specific database technology and is thus intended to work with any relational database management system. According to Johnson and Grandison, the Active Enforcement approach moves coding of fine-grained access control out of the application and into a common middleware layer. The middleware approach also provides the capability to integrate other security tools such as audit and intrusion detection capability (Agrawal et al, 2007). However, Johnson and Grandison did not discuss performance costs related to deploying Active Enforcement. As well, Roichman and Gudes (2007) noted that fine-grained access control implemented through a middleware layer is typically less effective than fine-grained access control implemented directly within the database.

Label Based Access Control

LBAC is a fine-grained access control implementation available in Oracle®, DB2®, and SQL Server® (Zhang, 2008). Corcoran et al. (2009) described LBAC in Oracle® and DB2® as "…lattice-ordered labels for implementing row-level security" (p. 280). LBAC is a mandatory access control implementation where security labels are employed to restrict the access of subjects to objects (Castano et al., 1994). LBAC is implemented within the database, providing a means to control access to sensitive data by tagging data with a data label (Oracle Corporation, 2009).

LBAC in DB2® enforces the policy that only a database security administrator may define a security policy (Bond et al., 2006). In DB2®, the role of the database security administrator is completely separate from the role of the database administrator (DBA) – this approach enforces separation of duties. With DB2®, security labels may be stored as arrays (ordered elements), sets (unordered elements), or as trees, using a hierarchical structure. Where security label hierarchies exist, the database enforces the access order.

Rask, Rubin, and Neumann (2005) described an LBAC implementation for the Microsoft SQL Server® relational database management system. According to Rask et al. (2005), fine-grained access control for SQL Server® is provided at the row-level using a view-based mechanism, which is implemented as follows:

1. Labels are assigned using standard role-based security;
2. Five dedicated system tables are provided for label management;
3. Views join base tables with system tables containing security labels;
4. Users may only access views – direct access to base tables is denied.

Fine-grained access control at the cell level is implemented in SQL Server® using encryption. According to Rask et al., a symmetric key is defined for each unique label; cells containing encrypted data must be accessed through a view that includes calls to decryption procedures stored in the database. As described by Rask et al., data in encrypted cells is only available to label holders with access to decryption keys. In SQL Server®, encryption and decryption is based upon the use of certificates and keys stored within the database. According to Rask et al., a thoroughly tested combination of views, database stored procedures, and triggers should be employed to manage the data encryption and decryption operations in order to ensure the security of user data.

Other Approaches to Fine-Grained Access Control

In addition to research that aligns with the view replacement model of fine-grained access control, a number of novel approaches to fine-grained access control have recently been investigated. Current research in this area falls into three broad categories: extensions to role-based access control, extensions to standard SQL, and use of encryption for fine-grained access control. However, regardless of the approach, Yu (2009) warns that providing fine-grained access control for modern database systems must also provide good performance.

Goyal, Pandey, Sahai, and Waters (2006) described a novel use of encryption to enforce fine-grained access control. In the scheme described by Goyal et al., data is stored in an encrypted form in conjunction with unencrypted security policies specifying the parties permitted to decrypt the data. However, in the work described by Goyal et al., only the performance implications of encryption/decryption methods are considered.

Performance considerations related to the complete end-to-end fine-grained access control solution were not evaluated.

Manjhi et al. (2006) examined the issue of query invalidation where the data underlying the query is encrypted and stored remotely by a database scalability service provider (DSSP). Under high demand conditions, Web-based applications can redirect read-only queries from the database directly linked to the application, to a database operated by the DSSP. According to Manjhi et al., a typical DSSP may store data from multiple customers in a single database, using data encryption to ensure that individual customer data remains private. In the DSSP model, updates are applied only against the database directly linked to the application (i.e., the master database, not the DSSP database). Thus, when the master database is updated, the read-only data cached by the DSSP must be invalidated and refreshed from the master. Of particular interest in the work reported by Manjhi et al. was the use of the TPC-W benchmark to estimate database scalability in a DSSP environment.

Tjan (2006) described an approach to database security that combined fine-grained access control and role-based access control. The approach described by Tjan required storage of both identity and access rules as part of the user data in order to facilitate fine-grained access control. According to Tjan, when injecting fine-grained access control information into the role-based access control system, the granularity of role-based access control was improved. At the same time, the security context payload associated with each row in a table could be reduced. Conflicts between the two access control mechanisms were resolved through rule ordering. Although simulation was used by Tjan to evaluate the efficiency of combining fine-grained access control and role-

based access control, results were presented in terms of complexity, and no specific performance metrics were provided.

Olson et al. (2008) described a novel approach to database fine-grained access control termed Reflective Database Access Control (RDBAC). The approach described by Olson et al. employed an approach conceptually similar to VPD in an Oracle® database. According to Olson et al., leveraging contextual information stored as part of end-user data is superior to the coarse-grained security hierarchy provided through standard SQL in relational database management systems. Olson et al. implemented a prototype system written in Prolog on a small-scale server, and evaluated the performance cost of RDBAC using query execution times as the base metric.

Chaudhuri et al., (2007) proposed an extension to SQL GRANT and REVOKE statements to allow predicates to be appended to the respective statements. For example, a WHERE clause appended to a SQL GRANT statement could provide a suitable mechanism for implementing row-level security. As well, the same SQL extensions proposed by Chaudhuri et al. could also provide attribute filtering at the column level, attribute filtering at the cell level, and fine-grained authorizations for database-stored procedures. The SQL extensions proposed by Chaudhuri et al. are conceptually similar to a strategy for fine-grained access control proposed by Agrawal et al., 2005. However, where Agrawal et al. proposed a novel approach, the proposal by Chaudhuri et al. was based upon simple extensions to standard SQL. Thus, the solution proposed by Chaudhuri et al. could be easily applied to both new and existing applications.

Majedi et al. (2009) proposed extending the SQL GRANT and REVOKE statements to allow optional references to privacy policies that implement restrictions

such as purpose, generalization, visibility, and retention. The proposal by Majedi et al. is conceptually similar to the proposal of Chaudhuri et al. (2007) although the mechanism to provide fine-grained access control differs significantly between the two proposals. While, both proposals are based upon extensions to standard SQL, the proposal by Majedi et al. potentially provides better granularity for privacy protection because of the greater flexibility provided by policy-based grammar. According to Majedi et al., social networks along with e-Business, e-Government, and e-Health would benefit from relatively simple extensions to standard SQL. However, the proposal Majedi et al. was restricted to a conceptual model; a prototype was not constructed and the performance implications of extending standard SQL to include policy-based grammar were not examined.

Pun et al. (2009) proposed extensions to standard SQL at the table level to support privacy policies in social networking applications. Similar to the proposal of Majedi et al. (2009), the extensions proposed by Pun et al. were based on privacy policy restrictions. However, Pun et al. proposed that the restrictions be implemented directly on physical tables rather than through SQL GRANT and REVOKE statements. Implementation at the table level would allow the same mechanism to be used for both DAC and MAC security models. According to Pun et al., most social networking sites, for example Facebook®, Flickr®, and MySpace®, delegate responsibility for enabling privacy controls to the user community, often resulting in minimal or inconsistent privacy protection. The approach suggested by Pun et al. is intended as a simple, cost-effective mechanism to enforce privacy policies at the system level. Pun et al. chose not evaluate the performance implications of their proposed extensions to standard SQL grammar.

Simpson (2008) examined the requirements for flexible fine-grained security in social networking environments like Facebook® and LinkedIn®. According to Simpson, there are three key privacy principles for social networking sites:

1. Data owners should be responsible for identifying privacy requirements;
2. Fine-grained security is required because the privacy requirements of data owners are difficult to predict in advance and may change over time;
3. Data owners should be provided with a listing of who has accessed their data.

According to Simpson, lack of control over data sharing in social networks should be addressed by enhancing user-controlled authorizations. Simpson modeled privacy policies for social networks using the mathematical language Z, which employs SQL like syntax and predicates. Qamar, Ledru, and Idani (2011) validated Simpson's approach, to security modeling by demonstrating the practicality of employing the Z language as a generalized tool for validating security models. However, Simpson also recognized that social and legal solutions, in addition to technical solutions, might be required to ensure adequate data privacy for social networking sites. Simpson did not address performance issues associated with his proposed approach to fine-grained authorizations for social networks.

Slaymaker, Power, Russell, and Simpson (2008), and He and Yang (2009) described policy-driven frameworks useful for secure collaboration in healthcare systems. Slaymaker et al., and He and Yang separately explored the use of Web services (i.e., a middleware approach) as a mechanism for enforcing fine-grained access control. However, their respective proposals were largely theoretical, and did not consider the performance characteristics of their respective approaches to fine-grained access control.

Abramov, Anson, Dahan, Shoval, and Sturm (2012) proposed the use of embedded security policies within the database for fine-grained access control. Abramov et al. (2012) employed model-based security patterns implementing RBAC but including fine-grained constraints to provide row-level security at both the organizational and application level. Abramov et al. described a prototype implementation of the model employing the Oracle® database and proprietary database stored procedures typically used to implement Oracle® VPD.

Benchmarking

According to Kalibera and Tuma (2006), "One of the principal approaches to evaluating performance is benchmarking, where the system under test executes a model task, called benchmark, and the observed performance is used for the evaluation" (p. 63). Jain (1991) in the classic text "The Art of Computer Systems Performance Analysis" defined benchmarking as the commonly applied process of acquiring measurements for use in comparing performance between systems. Jain described the benchmarking process as encompassing the execution of one or more workloads (also termed benchmarks) in order to measure system performance. Gray (1992) introduced the concept of a domain-specific benchmark. According to Gray, a domain-specific benchmark "…specifies a synthetic workload characterizing typical applications in that problem domain" (p. 3). As described by Jain, synthetic workloads are comprised of fixed data sets containing elements that are representative of real-world data. However, synthetic data sets may also be used as an approach to anonymize real-world data in order to provide privacy protection (Wilson & Rosen, 2003).

Figure 9 depicts the concept of a benchmark hierarchy as described by Menascé and Almeida (2001). According to Menascé and Almeida, a benchmark hierarchy consists of benchmarks of varying complexity that range from basic benchmarks to very complex, domain-specific benchmarks. Jain (1991) described two well-defined categories of domain-specific benchmarks – the TPC business focused benchmarks and the Standard Performance Evaluation Corporation (SPEC) scientific and engineering application benchmarks. These domain-specific benchmarks continue to be widely used today, although individual benchmarks within these two domains have evolved significantly since they were introduced in the 1990's.

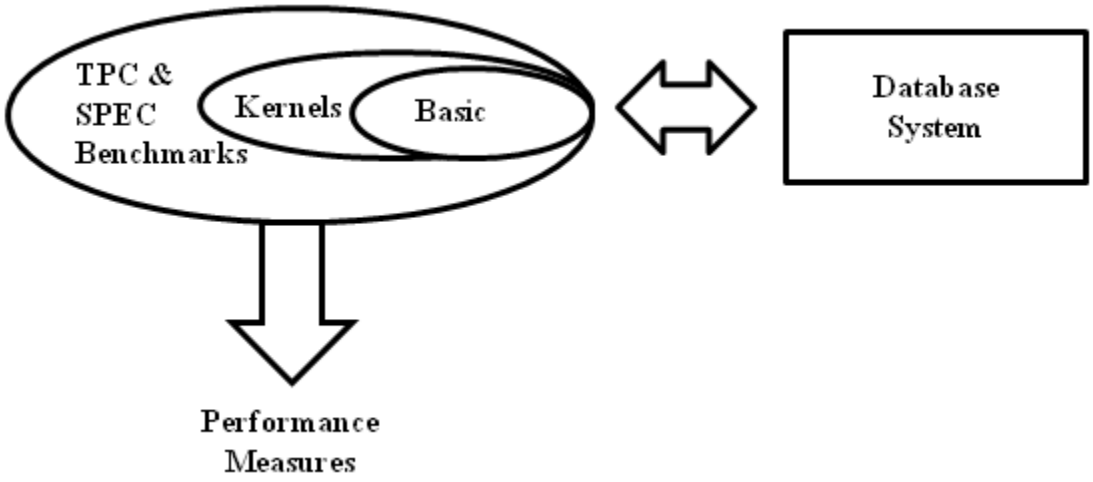

Figure 9. Benchmark Hierarchy (adapted from Menascé and Almeida, 2001, p. 266)

Formal benchmarks like TPC and SPEC are executed in tightly controlled environments and the results are subject to audit and verification prior to publication (Transaction Processing Performance Council, 1998). Carefully controlled and audited benchmark statistics are useful for comparing competing vendor products in the commercial environment (Gray, 1992). However, most commercial software license agreements contain clauses prohibiting publication of benchmark results that are obtained

without the explicit permission of the respective software vendor (Reed, 2006). Therefore, many researchers opt to use open source database software when conducting performance benchmarking in order to avoid potential software copyright violations or conflicts with software license agreements (Gonzalez, 2006).

A number of researchers have used the TPC benchmark to quantify the performance costs of fine-grained access control (Kabra et al., 2006; Zhu et al., 2006; Zhu & Lü, 2007; He & Veeraraghavan, 2009). However, the literature also contains a number of recent references to the legacy Wisconsin Benchmark (LeFevre et al., 2004; Wang et al., 2007; Zhu et al., 2008). Somewhat surprisingly, the Wisconsin Benchmark continues to be used in quantifying the performance aspects of fine-grained access control, in spite of the fact that it is does not adequately simulate a multi-user environment (Bitton, DeWitt, & Turbyfill, 1983). However, as noted by Elnikety et al. (2009), researchers frequently take advantage of readily available benchmarks like the TPC (or Wisconsin) benchmarks due to the ease of implementation and the generalizability of results. Figure 10 depicts a conceptual framework for benchmarking that encompasses three major components – a workload generator, the system under test (SUT), and a performance monitor (Menascé, 2002). The performance monitor depicted in Figure 10 is employed to measure the rate of arriving and completed requests processed by the SUT. According to Menascé, generated workloads may be classified according to two characteristics – type and intensity. In terms of generated database workloads, the classification of workload type refers to the nature of the database transaction – read, insert, update, or delete. In the database environment, workload intensity is typically expressed in terms of transactions per second (TPS). Kalibera and

Tuma (2006) discussed the effect of random fluctuations on benchmark results and identified four issues that may cause random fluctuations affecting benchmark performance:

1. System/database start-up;
2. Loading the benchmark into memory;
3. Measuring benchmark performance;
4. Caching in memory of benchmark intermediate results.

Kalibera and Tuma indicated that statistical averaging could be applied to summarize the results of benchmark testing. However, to employ statistical averaging, each step in the benchmark experiment must be executed multiple times and results collected must be averaged for each discrete step in the benchmark experiment (Kalibera and Tuma).

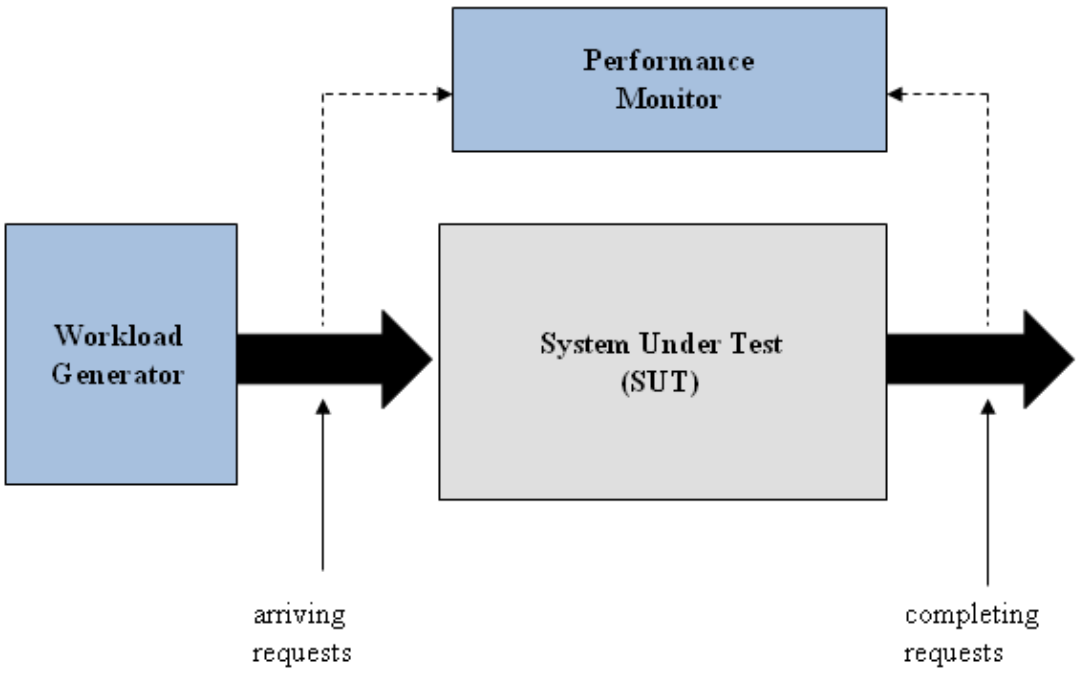

Figure 10. Conceptual Benchmarking Framework (adapted from Menascé, 2002, p. 84)

Kalibera, Bulej, and Tuma (2005) defined a statistic termed an "impact factor" that may affect benchmark results. According to Kalibera et al. (2005), it is not possible to define the initial state at the beginning of each benchmark experiment, thus subsequent benchmark runs are influenced to some degree by the presence of a random initial state. To address this problem, an impact factor may be calculated as "…the ratio of the standard deviation of samples from different benchmark runs compared to the standard deviation of samples from individual benchmark runs" (Kalibera et al., 2005, p. 855). According to Kalibera et al., an impact factor greater than one indicates a skewed benchmark result due to random perturbations affecting the initial state.

Kalibera and Tuma (2006) discussed the potential impact of memory caching on compilation times for open source packages and the resultant random skewing of benchmark results based upon package compile times. Similar issues have been noted in database systems. Bitton et al. (1983) provided recommendations for running database benchmarks in order to avoid skewing effects introduced by memory caching, including:

1. Table size should be five times greater than the physical size of the database buffer cache;
2. Response times should be computed as the mean of repeated queries;
3. Adjacent queries should be executed against separate tables to minimize the likelihood that the data requested resides in the database buffer cache;
4. Performance statistics should be based upon elapsed times to minimize confounding variations imposed by hardware and operating systems.

TPC Benchmarks

Research investigating the performance impacts of fine-grained access control frequently utilizes one of the TPC benchmarks for performance evaluations. A summary of three TPC benchmarks commonly used in quantifying fine-grained access control performance costs are provided in Table 1. The TPC benchmarks listed in Table 1 were developed to quantify the performance of specific workload types, specifically On-line Transaction Processing (OLTP), Decision Support Systems (DSS), and e-Commerce. The TPC-C (OLTP) benchmark was designed to measure performance characteristics associated with the insertion of new transactions (Transaction Processing Performance Council, 2010). According to Chakraborty, Majumdar, and Sural (2010), the TPC-C benchmark can be useful in security benchmarking as the transaction types and probability of occurrence are formally specified for the benchmark. The TPC-H (DSS) benchmark was designed to simulate ad hoc business queries against large, complex databases like data warehouses (Transaction Processing Performance Council). The TPC-W benchmark, intended to simulate a business-to-consumer (B2C) Web site, was designed to be scalable using modest sized databases and a fixed ratio of read and write transactions (Menascé, 2002), According to Menascé, the TPC-W benchmark "…tests many of the important elements of most e-commerce applications…" (p. 87).

Table 1. TPC Benchmarks (Transaction Performance Processing Council, 2003)

Benchmark	Type	Application Emulation	Initially Published
TPC-C	OLTP	On-Line Warehouse	July 1992
TPC-H	Decision Support	Data Warehouse	February 1999
TPC-W	Web e-Commerce	On-line Store	December 1999

According to Keeton and Patterson (2000), it is not generally feasible for researchers to run full-scale database benchmarks. Full-scale benchmarks in the

commercial environment, like the TPC benchmarks, typically required servers and associated hardware costing millions of dollars and involving complex configuration and tuning challenges (Keeton & Patterson). Instead, researchers often adapt existing benchmarks like the TPC benchmarks to run on small-scale servers (Elnikety et al., 2009). Alternatively, researcher may choose to use microbenchmarks, which are generally easier to implement and have modest hardware and software requirements (Keeton & Patterson). Microbenchmarks can also be used to investigate performance of isolated features of a system or application (Michiels, Manolescu, & Miachon, 2008).

Vieira and Madeira (2005) proposed the use of a metric called "security requirement fulfillment" (SRF) as a method for estimating the overhead imposed by database security using a scale ranging from zero to 100. According to Vieira and Madeira, the SRF metric is a composite rating for the efficiency of multiple security mechanisms including authentication, authorization, fine-grained access control, and encryption. The SRF value proposed by Vieira and Madeira is calculated by measuring the elapsed time of execution for a series of predefined read and update statements executing against the TPC-W benchmark tables. The SRF was proposed as a conceptual model for quantifying database performance when the database is constrained by one or more security mechanisms. Vieira and Madeira did not provide any experimental data to validate the SRF model.

Zhan et al. (2006) implemented an approach to fine-grained access control in the PostgreSQL 7.3 open source database. Performance of the approach chosen by Zhan et al. was evaluated using the TPC-C database benchmark. Results reported were based upon elapsed time of execution with a 90% confidence level. Zhan et al. reported 20-30%

degradation in performance using their novel fine-grained access control mechanism in the PostgreSQL 7.3 relational database management system.

Kabra et al. (2006) evaluated the performance aspects of a novel approach to query simplification on a Microsoft SQL Server® database using the TPC-H benchmark. Results reported were based upon elapsed time of execution. Kabra et al. reported that a 10-15% performance penalty was associated with implementation of their query simplification mechanism. However, Kabra et al., also found that the overhead imposed by query simplification was offset by significant improvements in query throughput. According to Kabra et al., redundancy removal is a practical approach to improving the efficiency of fine-grained access control implemented using a view replacement model.

Zhu and Lü (2007) described the use of security policies implemented as SQL statements to provide fine-grained access control in a relational database management system. The approach described by Zhu and Lü extends SQL capability in a manner that conceptually parallels the implementation of the Oracle® VPD technology by using database stored procedures and constraints to filter query results returned to the user. The goal of the work described by Zhu and Lü was to define a flexible, efficient framework for administering authorizations. Zhu and Lü evaluated the performance of their model using the TPC-W database benchmark executed against the DM5 relational database management system. Zhu and Lü reported a 10-15% performance penalty using their novel approach to fine-grained access control.

TPC-W Benchmark

The TPC-W benchmark suite has been used extensively for performance and security benchmarking (Vieira & Madeira, 2005; Kabra et al., 2006; Zhu & Lü, 2007;

Elnikety et al., 2009). As well, the TPC-W benchmark is routinely employed as a capacity-planning tool for e-commerce Web sites (Mi, Casale, Cherkasova, & Smirni, 2009). According to Menascé and Almeida (2001), the TPC-W benchmark is particularly useful in that it measures the performance of the whole system including CPU, network, I/O subsystem, operating system, and database. As well, source code for a number of TPC-W workload generators is freely available. For example, the University of Wisconsin distributes a Java-based TPC-W workload generator that is particularly useful to researchers (PHARM, n.d.).

The TPC-W benchmark was introduced in December 1999 and was designed to emulate an on-line bookseller (Transaction Processing Performance Council, 2003). The TPC-W benchmark measures the throughput of database transactions generated by a series of emulated web browsers. The metric termed "Web Interactions per Second" (WIPS) is used to report results for the TPC-W benchmark. According to the Transaction Processing Performance Council, the size of the database containing the on-line store is scaled based upon the number of rows in the ITEM table, which can range in size from 1,000 to 10,000,000 rows. The TPC-W benchmark is a popular choice for benchmark testing using small-scale systems given the simplicity of the benchmark structure and the small size of the base tables (Zhu & Lü, 2007; Elnikety et al., 2009; Mi et al., 2009).

According to Menascé (2002), scalability in the TPC-W benchmark is maintained by adhering to a fixed ratio or "scale factor" between the number of Web clients, termed "Emulated Browser Sessions" or EBS, and the cardinality of the ITEM table. The Transaction Processing Performance Council (2003) specifies that a minimum of 2,880 rows in the CUTOMER table must be configured for each EBS. Table 2 lists table

cardinalities for the TPC-W benchmark. According to the TPC-W benchmark guidelines published by the Transaction Processing Performance Council (2003), the cardinality of the ITEM table may range from 1,000 rows to 10,000,000 rows. The default cardinality for the ITEM table is 10,000 rows, which results in a database size of approximately 2.8 GB. However, Keeton and Patterson (2000) recommended that tables used for

Table 2. TPC-W Cardinalities (Transaction Performance Processing Council, 2003)

Table Name	Cardinality	Row Length	Table Size (MB)
CUSTOMER	2880 * (number of EBS)	760 bytes	258 MB
COUNTRY	92	70 bytes	0.006 MB
ADDRESS	2 * CUSTOMER	154 bytes	866 MB
ORDERS	.9 * CUSTOMER	220 bytes	557 MB
ORDER_LINE	3 * ORDERS	132 bytes	1002 MB
AUTHOR	.25 * ITEM	630 bytes	1.5 MB
CC_XACTS	1 * ORDERS	80 bytes	202 MB
ITEM	1K – 10M	860 bytes	8.3 MB

benchmarking should be at least four times the size of the database buffer cache and Bitton et al. (1983) provided a similar recommendation, namely that tables should be sized approximately five times larger than the database buffer cache. According to both Keeton and Patterson, and Bitton et al., these recommendations should be followed to ensure that SQL benchmark queries generate disk I/O requests, rather than retrieving data already resident in the database buffer cache. Therefore, the number of rows in the TPC-W ITEM table generally needs to be set higher than the default value of 10,000 rows in order to avoid the skewing effect sometimes evident when benchmark data is retrieved partially from the database buffer cache and partially from slower disk storage.

Elnikety et al. (2009) described the composition of the workload mixes in the TPC-W benchmark as follows:

- Browsing mix – 5% update transactions, 95% read-only transactions;

- Shopping mix – 20% update transactions, 80% read-only transactions;
- Ordering mix – 50% update transactions, 50% read-only transactions.

According to Elnikety et al., the shopping mix is the primary benchmark workload. The experimental design employed by Elnikety et al. demonstrated that the TPC-W workload mix scales linearly on small-scale database servers, due in part to the preponderance of read-only transactions in the combined workload mixes. Figure 11 depicts the logical

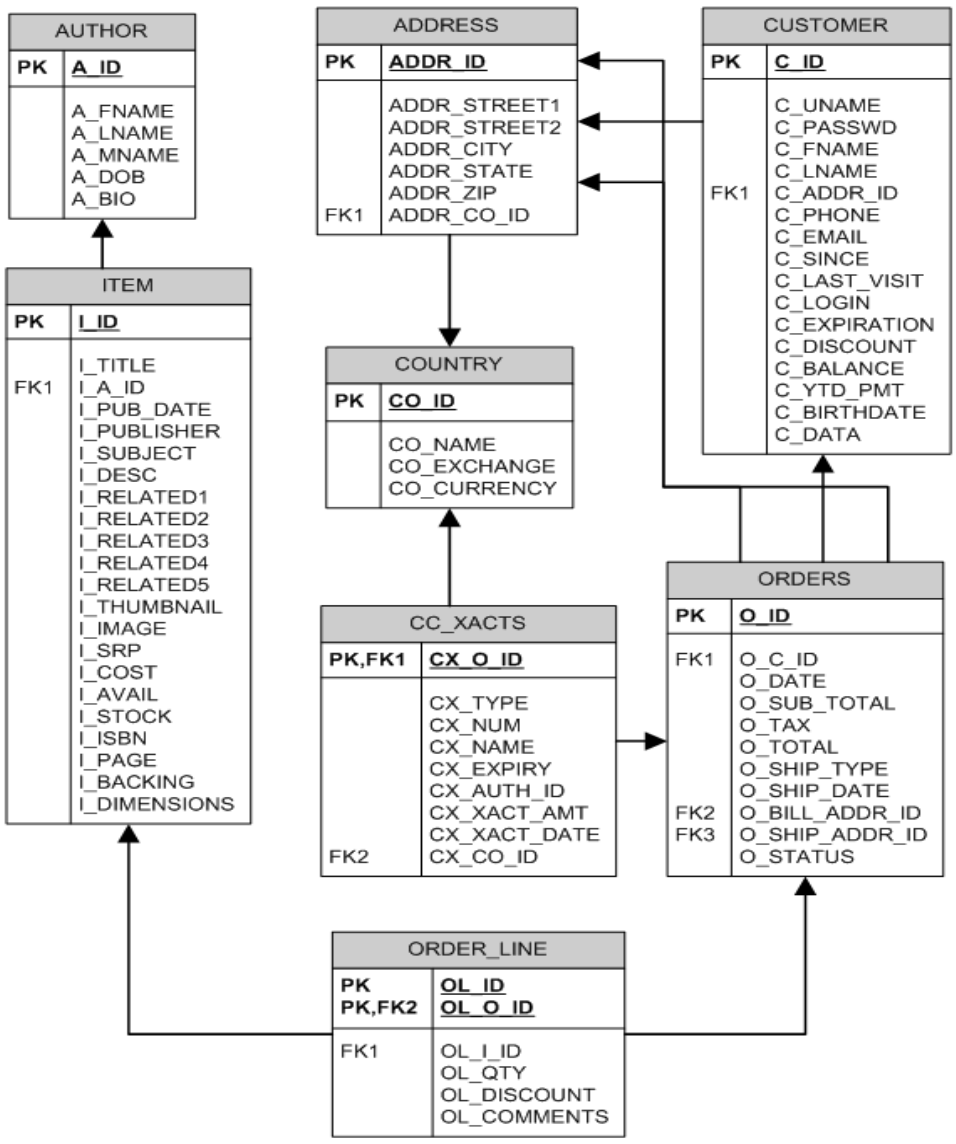

Figure 11. TPC-W Logical Data Model (adapted from Menascé, 2002, p. 85)

data model for the eight tables that comprise the database component of the TPC-W benchmark. The cardinality of ADDRESS, ODERS, ORDER_LINE, and CC_XACT are determined as a fixed ratio to the cardinality of the CUSTOMER table. The COUNTRY table contains a fixed number of rows; the AUTHOR table must be populated as fixed multiple (25%) of the size of the ITEM table (Transaction Processing Performance Council, 2003).

Performance Models

Jain (1991) stated that, "Queuing theory is the key analytical modeling technique used for computer systems performance analysis" (p. 505). According to Jain, simple systems, such as those defined in the TPC-W benchmark specification, can be represented using a network of several queues. Thus, queuing theory provides the theoretical foundation underlying benchmarks such as the TPC-W benchmark.

Queuing network models or systems are classified as open, mixed, or closed. In a closed queuing system, arrival of new jobs is constrained by the number of jobs currently being executed, whereas in an open system, arrival of new jobs is independent of the number of executing jobs (Schroeder, Wierman, & Harchol-Balter (2006). A mixed queuing network model combines elements of both open and closed system models. However, Schroeder et al. cautioned that open and closed systems could produce significantly different results using the identical benchmark executing against the same database. Thus, it is attendant upon researchers to both understand, and clearly describe the queuing model being employed when reporting experimental results derived from benchmarking.

The TPC-W database benchmark conforms to the closed network queuing model (Schroeder et al., 2006). According to Schroeder et al. (2006), the TPC-W benchmark specifies that a fixed number of database connections must be reserved for each Web client. Under the closed network queuing model, new connection requests are queued if free database connections are not available.

Elnikety et al. (2009) described an approach for predicting database workload scalability using a closed queuing network model. The model proposed by Elnikety et al. utilized data collected from the TPC-W benchmark. According to Menascé (2002), the TPC-W benchmark is well suited for this task as it can be used to measure both throughput and scalability. Zhu et al. (2006) and Zhu and Lü (2007) effectively employed the TPC-W benchmark to evaluate the performance impacts of fine-grained access control implementations in relational database management systems by evaluating changes in throughput and scalability in the presence of fine-grained access control.

Jin, Tan, Han, and Liu (2007) cautioned that the use of performance models such as queuing models and Marchov chain models are best suited for estimating the performance of new systems under design, and are potentially unsuited for modeling systems already in production. Specifically, Jin et al. (2007) stated that extrapolating performance of legacy relational database management systems, where database activity is projected to increase substantially, requires detailed knowledge of the application and the operating environment. According to Jin et al., “To make use of them [analytical models], one needs to understand and quantify the system’s actual performance characteristics and operating environment” (p. 540).

The Contribution This Study Will Make to the Field

There is relatively little data in the literature quantifying the performance of database fine-grained access control (Zhang, 2008). Yet, in terms of real-world applications, there is a genuine and increasing need for the use of database fine-grained access control, particularly for Web-based applications (Zhu et al., 2008). This study was undertaken to address the question "Which approach to providing database fine-grained access control offers better performance?" As well, the study also examined the issue of scalability for database fine-grained access control. Four fine-grained access control mechanisms for relational database systems were evaluated using security benchmarking – authorization views, transparent query rewrite, a Hippocratic Database, and label-based access control. Using metrics derived from security benchmarking, a generic model was developed. The model can be used to understand the performance and scalability impacts of the four selected fine-grained access control mechanisms. The model provides a research-based tool that should help to dispel concerns about the performance and scalability of fine-grained access control for relational database management systems.

Summary

Roichman and Gudes (2007) and Zhu and Lü (2007) identified the integration of database systems with Internet technology as driving the requirement for database fine-grained access control in order to ensure suitable data security. Johnson and Grandison (2007) identified fine-grained access as a suitable mechanism to address legislated requirements for data privacy and to cope with the complexities of cross-jurisdictional data privacy requirements. Yet, a standard approach to implementing fine-grained access

control for relational database management systems remains elusive. Wang et al. (2007) cited the technical difficulties as the most critical factor limiting the widespread deployment of fine-grained access control in relational database management systems. Further, Bertino and Sandhu (2005) noted that all of the current approaches to implementing fine-grained access control in relational database management systems are problematic, and new solutions are potentially required. Specifically, Bertino and Sandhu noted that most of the current approaches to fine-grained access control based upon view replacement and thus can return incorrect results or exhibit decidability problems.

Nevertheless, commercial database products that support fine-grained access control are available from IBM®, Oracle®, Sybase® and Microsoft® (Kabra et al., 2006; Zhang, 2008). In addition, generic approaches to database fine-grained access control such as Hippocratic Databases and authorization views offer feasible and pragmatic alternatives to commercial products (Bertino et al., 2005). Agrawal et al. (2005) identified simplicity and ease of maintenance as two of the most important factors in selecting a fine-grained access control technology for relational database management systems. However, performance is also an important criterion in selecting an appropriate fine-grained access control mechanism, although performance is often treated as a secondary consideration (Kabra et al., 2006).

The approach to database fine-grained access control described by Abramov et al. (2012) reflects a trend evident in recent research related to the use of fine-grained access control. Recent research in this area tends to focus upon the use of access control policies as a general mechanism for specification of fine-grained security, in both software systems and relational database management systems, without particular concern for the

efficiency of the mechanism. This trend towards generalization of fine-grained access control can be seen, for example, in the works of Krishnamurthy et al. (2010), Moore (2011), Abramov et al. (2012), and Wang et al. (2012).

At present, there are no known references in the literature describing comparative performance data for database fine-grained access control. In practice, security benchmarking can be employed for individual implementations of database fine-grained access control to provide comparative performance data for queries both with and without fine-grained access control. However, benchmarking is a complex and time-consuming process and may require expert knowledge for correct interpretation (Keeton & Paterson, 2000; Gunther, 2004). Analytical models, based upon data collected from benchmarking, provide a means to predict database performance and scalability using small-scale servers readily available to researchers and software developers.

Chapter 3

Methodology

Approach

The purpose of the author's study was to quantify the relationship between the use of database fine-grained access control and the resultant performance impact on database transaction throughput. Experiment is a research methodology employed to investigate causal relationships using controlled tests (Dawson, 2000). The cause-and-effect relationship between fine-grained access control and database performance was demonstrated through experiments conducted in the laboratory. Sekaran (2003) suggested that cause-and-effect relationships are best established through laboratory experiments. The study conducted confirmed a statistically significant cause-and-effect relationship between degradation in query performance and the use of fine-grained access control in a relational database management system.

Benchmarking was employed as the primary tool to quantify database performance at the transactional level. Benchmarking provides a means to measure efficiencies while simultaneously executing workloads representative of real-world conditions (Santos & Bernardino, 2009). While primarily viewed as a tool for comparing the relative performance of different server hardware, benchmarking is also an effective

tool for comparing the performance of similar or competing software products (Gray, 1992). Moreover, benchmarking provides a practical approach to compare database security mechanisms between different database products (Vieira & Madeira, 2005).

The goal of the author's study was to develop and validate a generic performance model for Fine-grained Access Control Evaluation (FACE). The FACE model is considered generic in the sense that it is not restricted to any specific database product or server architecture (Melnik et al., 2003). The FACE model provides a high-level, architecture independent viewpoint that quantifies the performance costs and scalability of database fine-grained access control. Metrics underlying the FACE model were derived from security performance benchmarking.

Benchmarking as an approach to quantifying performance costs of database fine-grained access control has been described in the literature in considerable detail by LeFevre et al., 2004, Kabra et al., 2006, Wang et al., 2007, Zhu et al., 2008, and Shi et al., (2009). Using the Wisconsin Benchmark and tables of one million, five million, 10 million, and 15 million rows, LeFevre et al. quantified query performance based upon elapsed time of execution for queries both with and without fine-grained access control. Kabra et al. evaluated the performance of selected TPC-H benchmark queries; those queries were re-written to remove redundancies introduced by the addition of fine-grained access control predicates. Kabra et al. quantified the performance of modified and unmodified TPC-H queries by measuring query execution time. Wang et al. implemented a fine-grained access control scheme using SQL CASE statements against tables from the Wisconsin benchmark; performance was quantified by measuring query execution time for modified and unmodified queries against a table containing one

hundred thousand rows. Zhu et al. employed SQL CASE statements and the SQL MINUS operator to modify SQL queries in order to mask data and to implement security polices termed “enforcing rules”. Zhu et al. quantified the performance of unmodified queries, row-level access control enforcement, and cell-level enforcement through comparing the respective query execution times against a table in the Wisconsin benchmark containing one hundred thousand rows. In an approach conceptually similar to Zhu et al., Shi et al. used a combination of authorization views, SQL CASE statements, nullification, and the SQL MINUS operator to enforce row and cell-level access control. Shi et al. quantified query performance using elapsed time of execution against a table containing one hundred thousand rows derived from the Wisconsin benchmark.

According to DeWitt (1992) in his seminal paper describing the history of the Wisconsin Benchmark, the elapsed time of execution is the best metric for measuring database transactional throughput. Dewitt also asserted that elapsed time of execution can be measured consistently between databases of different architectures, using disparate server hardware, and different operating systems. Modern database benchmarks continue to be based upon this approach to benchmarking. The FACE model quantifies the performance and scalability of four commonly employed fine-grained access control mechanisms – authorization views, the Hippocratic database, LBAC, and transparent query rewrite. The FACE model was derived using the following approach:

- Evaluation of multiple fine-grained access control mechanisms using the same controlled hardware and software environment;
- Evaluation of multiple fine-grained access control mechanisms using both synthetic and real-world data;

- Evaluation of fine-grained access control mechanisms using complex transactions based upon queries that differ by an order of magnitude in complexity (as determined by query cost analysis).

The FACE model provides a simple visual model to represent the performance of multiple fine-grained access control mechanisms evaluated in the same controlled database environment. The objective in formulating the FACE model was to provide an answer to the question "Which fine-grained access control mechanism offers the better performance and scalability?" In many cases, database professionals and system architects have limited control or influence over the specific database technology used when developing a new application (Rosenthal, Sciore, and Doshi, 1999). However, they often do have a significant degree of control over database security decisions, including whether or not to implement database fine-grained access control.

Data

Two data sets were utilized in the derivation of the FACE model. One of the data sets contained synthetic data as defined by the Transaction Processing Performance Council (2003) for the TPC-W benchmark. The other data set contained real-world data extracted from a large wildlife habitat capability and suitability model. Each data set was evaluated independently, both with and without the use of database fine-grained access control.

The TPC-W benchmark data was used as the basis for formulating the FACE model. Elnikety et al. (2009) demonstrated that the TPC-W benchmark is particularly well suited for use in evaluating database performance and scalability on small-scale database servers. As well, Mi et al. (2009) described the TPC-W as a credible benchmark

that is routinely used for evaluating both middle-tier and database performance. It should be noted however that the TPC-W benchmark was declared obsolete as a commercial benchmark as of April 28, 2005 (Transaction Processing Performance Council, n.d.). Nevertheless, the TPC-W benchmark continues to be widely used in academic research. The literature contains numerous references to the TPC-W benchmark methodology as well as a significant number of references citing database performance metrics derived from benchmarking using the TPC-W benchmark (Mi et al., 2009).

The wildlife habitat capability and suitability data set was used to validate the FACE model using a series of custom benchmark queries; a sample of these queries is provided in Appendix F. Database performance and scalability were evaluated using this data set both with and without fine-grained access control. A Web application was developed in 2002 to access the wildlife habitat capability and suitability data as part of a project to make summary wildlife capability and suitability data accessible to the public. The Web application included a number of complex SQL queries that incorporated joins over the two largest tables – the capability table and the suitability table. The SQL code for these complex queries was extracted from the application code and was adapted to provide representative benchmarks. Given the processing intensive nature of these queries, they were considered good candidates for use in evaluating the performance overhead imposed by fine-grained access control.

The wildlife habitat capability and suitability data set contains two large tables, each containing 32 million rows, and a small number of related support tables. A logical data model for the wildlife habitat capability and suitability application data is provided in Figure 12. In March 2009, the wildlife habitat capability and suitability Web

application was taken offline. The application was infrequently used and no further funding was available for application maintenance. Thus, the Web application is no longer available to the public.

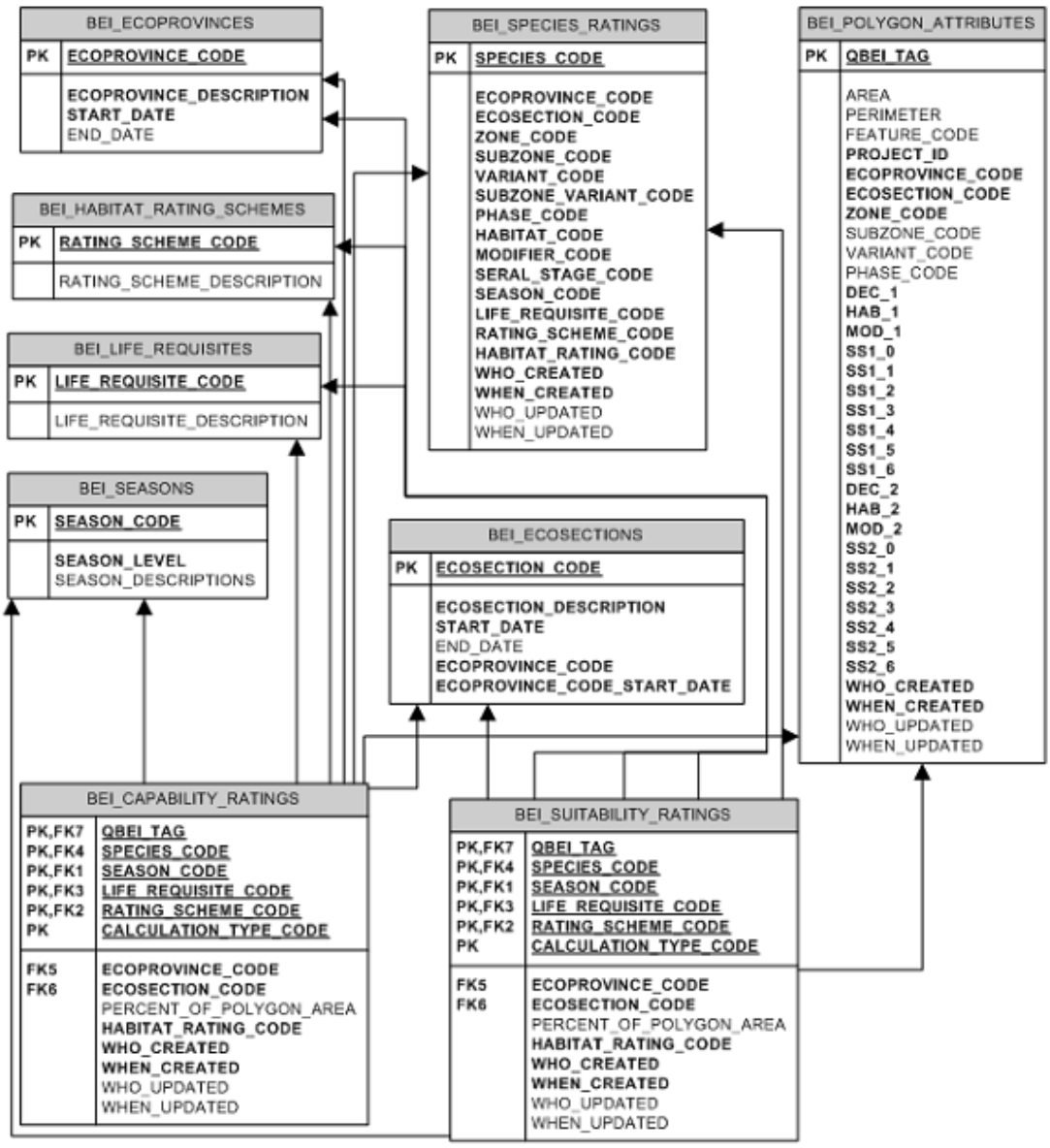

Figure 12. Habitat Capability-Suitability Logical Data Model

The TPC-W database tables, and the wildlife habitat capability and suitability database tables were modified to add three additional columns that were used as the basis

of privacy enforcement in a Hippocratic Database configuration following the approach described by LeFevre et al. (2004). As well, a column containing a security label was added to each table to support evaluation of fine-grained access control mechanisms using authorization views, transparent query rewrite, and LBAC. Similar modifications were made to the structure of the TPC-W benchmark tables. LeFevre et al. evaluated the impact of modifying database tables to include privacy enforcement data and concluded that the impact on performance was minimal when the number of stored choices is kept to a minimum.

Experimental Design

The FACE model provides a simple means to quantify the performance costs of fine-grained access control. Past research has identified a number of variables that have the potential to constrain the performance of relational database management systems (DeWitt, 1992; Keeton & Patterson, 2000; LeFevre et al., 2004; Avritzerv& Weyuker, 2004; Kabra et al., 2006; Chaudhuri, 2009; Zhang et al., 2009). In the author's study, the effect on database performance of three variables was measured through experimentation for the TPC-W data set. The three variables are described in Table 3. However, only the first two variables, fine-grained access control and query complexity were evaluated for the wildlife data set. The wildlife application, from which sample SELECT benchmark queries were extracted, contained no SQL INSERT or UPDATE statements.

Table 3. Experiment Variables and Levels

Symbol	Variable	Level -1	Level +1
A	Fine-grained Access Control	Not Enabled	Enabled
B	Query Complexity	Simple	Complex
C	SQL Operation	Select	Update

The procedure for measuring experimental effects was as follows:

1. Independent Variable
    - The experimental design evaluated the effect of fine-grained access control on database performance;
    - The independent variable was evaluated at two levels – not enabled (-1), and enabled (+1).
2. Dependent Variables
    - The experimental design evaluated the effect of two dependent variables, query complexity, and SQL operation, on the independent variable;
    - Each dependent variable had two levels, represented by (-1) and (+1);
    - Query complexity was estimated based upon cost analysis – simple queries were assigned a value of -1 and complex queries were assigned a value of +1;
    - Two SQL operations were evaluated – SELECT operations were assigned a value of -1 and UPDATE operations were assigned a value of +1.

Zhu and Lü (2007) evaluated their fine-grained access control model using a subset of the TPC-W benchmark. Specifically, Zhu and Lü employed queries against the ORDER and ORDER_LINE tables to evaluate performance under load. In the experimental design employed by the author, a subset of the TPC-W benchmark, consisting of two SELECT queries and two UPDATE queries was used to quantify performance.

Table 4 lists the queries employed in the evaluation. Representative queries were selected based on estimated query cost and the desire to avoid table access conflicts.

Table 4. Measures of complexity for selected TPC-W queries

Query Name	Estimated Cost	SQL Operation	Query Complexity
getName	16.62	SELECT	Simple
getBook	33.23	SELECT	Complex
resetCartTime	6.65	UPDATE	Simple
refreshCartUpdate	33.30	UPDATE	Complex

Both LeFevre et al. (2004) and Kabra et al. (2006) warned that implementation of fine-grained access control in relational database management systems typically results in increased complexity of SQL queries. Kabra et al. and Chaudhuri (2009) documented that increased complexity of SQL queries leads to increased query optimization times and longer elapsed times for query execution. Thus, given the known relationship between query complexity and query performance, a metric representing query complexity was one of the independent variables employed in the author's experimental design.

Most of the current literature describes the performance implications of fine-grained access control only as they relate to read-only transactions. Wang et al. (2007) stated that fine-grained authorizations are more likely to be applied to read-only (i.e., SQL SELECT) operations than to update or delete operations. Kocatürk and Gündem (2008) justified a focus on read-only operations on the basis that the majority of users accessing database tables are limited to read-only access. In addition, Corcoran et al. (2009) observed that update operations that occur simultaneously with the execution of read-only queries could inhibit effective query optimization. Zhang et al. (2009) noted that in a mixed transaction environment (i.e., including both read and insert/delete operations); periodic resampling of table cardinality is suggested if effective query optimization is to be achieved. To address the issues raised by Kocatürk and Gündem, and Corcoran et al., the experimental design employed by the author evaluated read and

update operations separately. To address the query optimization issue raised by Zhang et al., table cardinality remained fixed throughout the benchmarking process.

A 2^k factorial design can be used to measure the effect of k variables, where each variable has two levels (Jain, 1991). A $2^k r$ factorial design extends the 2^k factorial design by including r replications of each experiment to allow experimental error to be calculated. A $2^k r$ factorial design was used to structure the experiments conducted by the author. Table 3 describes the variables that were measured in the experiments – each variable was evaluated at two levels. According to Jain, "The variation due to a factor [variable] must be compared with that due to errors before making a decision about its effect" (p. 278).

Specific Procedures Employed

The author evaluated four different implementation models of database fine-grained access control against three different databases, containing an ITEM table with one hundred thousand, one million, and 10 million rows respectively. According to Menascé (2002), scalability in the TPC-W benchmark can be established by varying the cardinality of the ITEM table. The implementation models evaluated by the author included:

1. Simple parameterized authorization views based on user-id as described by Rizvi et al. (2004) and as shown in the SQL code listing in Appendix A;
2. A prototype Hippocratic Database based upon the methodology described by LeFevre et al. (2004) and as shown in the SQL code listing in Appendix B;
3. An implementation of LBAC similar to the approach described by Rask et al. (2005) for SQL Server® and as shown in the SQL code listing in Appendix C;

4. An implementation of transparent query rewrite conceptually similar to Oracle® VPD. Transparent query rewrite was implemented under PostgreSQL using views and a database stored procedure to fetch the user security context from memory. SQL code for the views is provided in Appendix D.

While the implementation models chosen for evaluation do not embody an exhaustive list of fine-grained access control mechanisms, they are considered representative of past research in the field (Zhang, 2008). In addition, the implementation models that were chosen all have a well-documented theoretical foundation (Bertino et al., 2005). Each of the chosen implementations of fine-grained access control was evaluated using the PostgreSQL open source database. PostgreSQL is a popular open source relational database management system that runs on multiple platforms including Windows, Linux, and UNIX. PostgreSQL is the successor to the INGRES® database system, first used to demonstrate fine-grained access control through query rewriting (Stonebraker & Rowe, 1986).

The experimental set-up as depicted in Figure 13 consisted of a workload generator and a dedicated database server. The workload generator and database server were interconnected using a dedicated Gigabit network isolated from Internet traffic. Query performance was measured based upon the elapsed time to completion for each transaction. The author's experimental set-up was similar to experimental hardware and software configurations described by Zhu et al. (2006), Elnikety et al. (2009), and Mi et al. (2009).

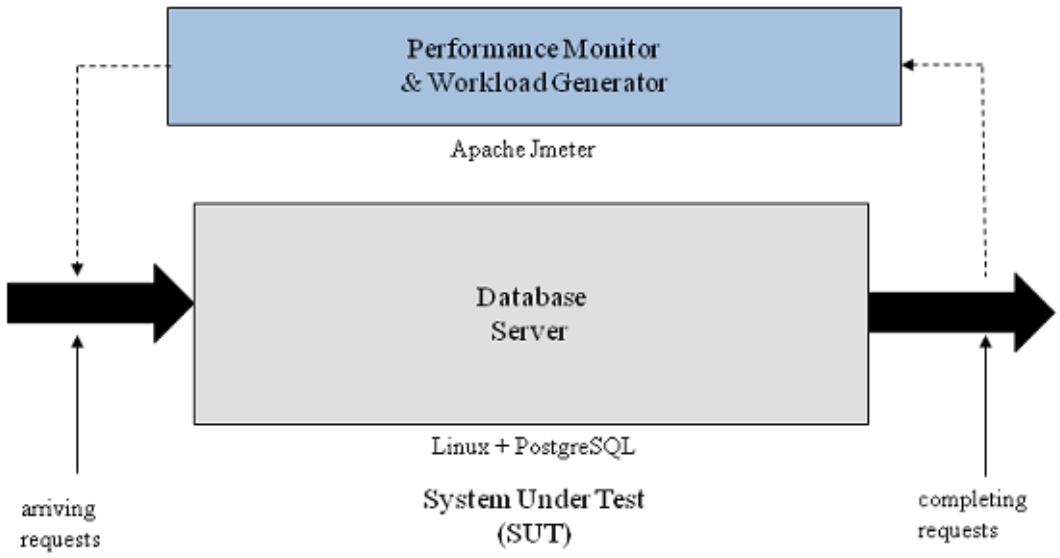

Figure 13. Physical Framework for Benchmarking

Generalizability was an important consideration in developing the FACE model. The model must be theoretically sound but must also be relevant in providing a basis to extrapolate the performance and scalability of real-world database implementations. According to Sekaran (2003), experimental results that can be verified through field-testing are useful in establishing generalizability. Employing an open source database and the TPC-W benchmark in the development of the FACE model was intended to ensure that the model is generalizable and that other researchers can verify published results.

Table 5. Benchmark Test Bed Environment

Resource	Client	Database Server
OS	Windows 2008 R2	CentOS 5.6
Application	Jmeter	PostgreSQL 9.0.4
CPU	2 X 2.8 GHz AMD	1 X 2.13 GHz AMD
Memory	8 GB RAM	2 GB RAM
Disk	320 GB SATA	80 GB IDE/160 GB SATA
Network	1 Gbit	1 Gbit

Table 5 provides a summary of the hardware and software environment that comprised the benchmark test bed employed in the experimentation portion of the

author's study. The Apache Jmeter open source test tool was used to manage the TPC-W queries and to record query execution times. According to the Apache Software Foundation (n.d.), the Jmeter tool provides the capability to simulate multiple user access by allowing multiple concurrent threads of execution to be managed through Jmeter.

Keeton and Patterson (2000) noted that simulation using small-scale processors is a pragmatic approach to modeling performance of large-scale systems. According to Keeton and Patterson, large-scale systems may be comprised of multiple large processors, many gigabytes of physical memory, and hundreds of disk storage devices. Thus, only researchers with commercial ties are likely to undertake performance evaluations using large-scale systems (Keeton & Patterson).

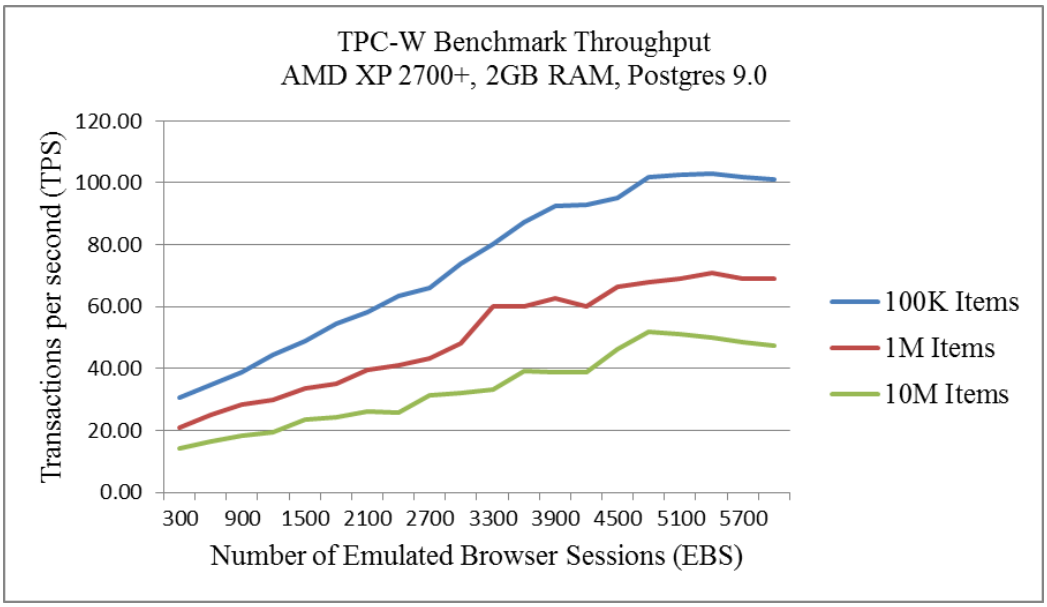

Figure 14. TPC-W Benchmark Scalability on a Small-Scale Processor

Menascé (2002) described the TPC-W benchmark as a scalable benchmark. Scalability in the TPC-W benchmark may be demonstrated by varying either the cardinality of the ITEM table, the number of EBS's, or both (Menascé). Figure 14 depicts

TPC-W benchmark results obtained by the author using the getBestSellers query evaluated against the ITEM table containing one hundred thousand, one million, and 10 million rows respectively. As well, the number of EBSs was incremented from three hundred to 6000. Figure 14 depicts classical canonical throughput characteristics (throughput versus load) for each of the three respective workloads (Gunther, 2004).

Phase 1 – Baseline Performance Evaluation using the TPC-W Benchmark

In the first phase of the author's experimentation, the test bed environment was configured and baseline results were collected. Specific procedures undertaken included:

1. Configuring the servers and database software;
2. Populating three shared PostgreSQL databases with the TPC-W benchmark data; Modifying the TPC-W base tables to include additional columns to be used in the implementation of fine-grained access control;
3. Creating four additional database schemas per database, one for each implementation of fine-grained access control to be evaluated; access control mechanisms implemented within these schemas utilized data from the common (public) database schema through the use of views;
4. Configuring the Jmeter test tool to launch and monitor TPC-W workloads;
5. Executing three trials of each TPC-W workload without fine-grained access control (i.e., the null case) to validate the test bed configuration. The database was restarted before each trial to flush both the database and operating system buffer caches. According to Smith (2007), the PostgreSQL database has a limited internal buffer cache and thus relies heavily upon the much larger operating system file cache as an extension to the internal database buffer cache.

Phase 2 – Performance Evaluation using the TPC-W Benchmark

In the second phase of the author's experimental work, three experiments were conducted to evaluate the performance of the selected fine-grained access control mechanisms against three separate databases, scaled against one hundred thousand items, one million items, and 10 million items respectively. Within each of the three experiments, five separate workloads were evaluated, ranging from 800 to 4000 EBS's. Each experiment employed a $2^k r$ factorial design using the TPC-W benchmark data to evaluate performance. Specific procedures undertaken included:

1. Three repetitions for each of the five TPC-W workloads executed against the authorization view schema;
2. Three repetitions for each of the five TPC-W workloads executed against the Hippocratic Database schema;
3. Three repetitions for each of the five TPC-W workloads against the LBAC schema;
4. Three repetitions for each of the five TPC-W workloads executed against the transparent query rewrite schema.

Phase 3 – Performance Evaluation of a Real-World Data Set

In the third phase of the author's experimental work, one additional experiment was conducted to evaluate the performance of the selected fine-grained access control mechanisms against the wildlife data set, which resided in a separate PostgreSQL database. Five separate workloads were evaluated, ranging from 800 to 4000 EBS's. The experiment employed a $2^k r$ factorial design using wildlife capability and suitability data to evaluate performance. Specific procedures undertaken included:

1. Populating a PostgreSQL databases with wildlife capability and suitability data;
2. Modifying the capability and suitability base tables to include additional columns to be used in the implementation of fine-grained access control;
3. Creating four additional database schemas, one for each implementation of fine-grained access control that was evaluated; access control mechanisms implemented within separate schemas accessed data from a shared schema;
4. Identifying SQL queries to be employed as workloads for evaluating each type of fine-grained access control against the wildlife data; the queries were derived from the legacy Web application used by the public to query wildlife data;
5. Configuring the Jmeter test tool to launch and monitor wildlife workloads;
6. Three repetitions for each of the five wildlife workloads executed against the shared schema containing the wildlife data base tables;
7. Three repetitions for each of the five wildlife workloads executed against the authorization view schema;
8. Three repetitions for each of the five wildlife workloads executed against the Hippocratic database schema;
9. Three repetitions for each of the five wildlife workloads executed against the LBAC schema;
10. Three repetitions for each of the five wildlife workloads executed against the transparent query rewrite schema.

Formats for Presenting Results

The results for each of the author's experiments are presented using sign tables. Table 6 provides an example of a sign table for a $2^k r$ experimental design employing

three repetitions (i.e., *r*=3). Jain (1991) describes the elements of a sign table for a $2^k r$ experimental design with three replications as follows:

- the columns A, B, and C represent variables in the experiment (i.e., *k*=3);
- each variable has two levels, indicated by the values 1 or -1;
- columns AB, AC, BC, and ABC represent the interaction between variables;
- data collected for each repetition is recorded in the columns y_1, y_2,y_3;
- the column Mean ȳ is the mean value of y_1, y_2,y_3 for the respective row;
- the product of the column values for a row and the mean value for the row (i.e., Mean ȳ) and are summed to derive the value "Total";
- columns labeled "Total/8" contains the column totals divided by the number of rows in the table; these values are represented by q_0, q_A, q_B,…q_{ABC}.

Table 6. Sign Table for a $2^3$3 Experimental Design (Adapted from Jain, 1991).

I	A	B	C	AB	AC	BC	ABC	y1	y2	y3	Mean ÿ
1	-1	-1	-1	1	1	1	-1	14	16	12	14
1	1	-1	-1	-1	-1	1	1	22	18	20	20
1	-1	1	-1	-1	1	-1	1	11	15	19	15
1	1	1	-1	1	-1	-1	-1	34	30	35	33
1	-1	-1	1	1	-1	-1	1	46	42	44	44
1	1	-1	1	-1	1	-1	-1	58	62	60	60
1	-1	1	1	-1	-1	1	-1	50	55	54	53
1	1	1	1	1	1	1	1	84	80	82	82
321	69	45	157	25	21	17	1	Total			
q0	qA	qB	qC	qAB	qAC	qBC	qABC				
40.125	8.625	5.625	19.625	3.125	2.625	2.125	0.125	Total/8			

According to Jain (1991), the results from a sign table for a 2^3r experimental design with three repetitions (*r*=3) can be analyzed using the following calculations:

Equation 1.1 $SSY = x_1^2 + y_2^2 + y_3^2 \ldots + y_x^2$ (Sum of squares of the observations)

Equation 1.2 $SS0 = 2^k r q_0^2$ (Sum of squares of the mean)

Equation 1.3 $SST = SSY - SS0$ (Sum of squares total)

The variables and their interactions in a $2^3$3 experimental design are represented by A, B,

C, AB, AC, BC, and ABC. The respective sums of squares are SSA, SSB, SSC, SSAB, SSAC, SSBC, and SSABC. The sum of squares for variable A is calculated as:

Equation 1.4 $SSA = 2^k r q_A^2$ (Sum of squares for *A*)

The percentage of variation attributed to variable A is calculated as:

Equation 1.5 $SSA / SST * 100\%$ (Percentage of variation due to *A*)

The sum of squares errors (*SSE*) can be calculated as:

Equation 1.6 $SSE = SSY - 2^3 r(q_0^2 + q_A^2 + q_B^2 + q_C^2 + q^2_{AB} + q^2_{AC} + q^2_{BC} + q^2_{ABC})$

Equation 1.7 $df = 2^3(r\text{-}1)$ (degrees of freedom for *SSE*)

Equation 1.8 $s_e = \sqrt{\frac{SSE}{2^3(r-1)}}$ (standard deviation of errors)

Equation 1.9 $s_{qi} = \frac{se}{\sqrt{2^3 r}}$ (standard deviation of effects)

Using the standard deviation of effects ($s_{qi} = 0.51$) from Equation 1.9 (based upon the sample data in Table 6) and the *t*-value from a standard *t*-table for 16 degrees of freedom and 90% confidence (i.e., $t_{(.10,16)}$=1.337), confidence intervals may be calculated as:

Equation 1.10 $q_i \mp 0.51*1.337$ (confidence intervals for q_i)

The value q_i represents the confidence intervals for the effects as measured for a specific sign table. Using q_i the confidence interval for the interaction between each of the experimental variables, A, B, C, AB, AC, BC, and ABC can be determined. For example, using the sample data in Table 6, the confidence intervals for q_A = (7.94, 9.31).

The most effective and recognized method to represent performance data is through derivation of a single numeric value or metric. Dewitt (1992) cautioned that performance data collected as part of academic research may be largely ignored unless it can be represented simply. The FACE model is presented as a simple series of line graphs

to allow easy interpretation. The novel aspect of the FACE model is that four different implementations of fine-grained access control are presented in the same graph based upon comparable measurements from a common test bed platform. Elnikety et al. (2009) used line graphs to depict performance and scalability separately – performance was depicted as a measure of transaction throughput and scalability was as a measure of transaction response time. The FACE model employs separate line graphs to depict performance and scalability following the methodology of Elnikety et al.

Resource Requirements

The resources required to conduct the author's study were modest. A single small-scale server was dedicated to hosting a PostgreSQL database. All of the application software required, CentOS Linux, the PostgreSQL database, and the Apache Jmeter test tool are freely available as open source software to any researcher. Response time metrics were collected using the Jmeter open source tool from the Apache Software Foundation. Experimental data was analyzed in Microsoft Excel following the sign table methodology for calculating multi-variable effects as described by Jain (1991). The TPC-W benchmark data was generated using the PHARM (n.d.) distribution from the University of Wisconsin. The British Columbia Ministry of Environment provided the real-world wildlife habitat capability and suitability data used in this project. While the wildlife habitat capability and suitability data are in the public domain, only response times for queries against the wildlife habitat capability and suitability data are presented for use in validating the FACE model. The wildlife habitat capability and suitability data was

formerly deployed in a publically accessible Web-based application that did not include any read-write transactions.

Reliability and Validity

According to Salkind (2006), reliability indicates the consistency, stability, and predictability of the measurements in an experiment. Sekaran (2003) describes two forms of validity of concern to experimental design – internal validity and external validity. According to Sekaran, internal validity is the degree of authenticity in a cause-and-effect relationship whereas external validity constitutes the degree to which results can be generalized to real-world situations.

In the experiments conducted, reliability was addressed through the use of commodity hardware and open source software, thereby allowing the experimental set-up to be replicated by other researchers. The experimental set-up employed methodologies similar to those described by Keeton and Patterson (2000), LeFevre et al., (2004), Zhu and Lü (2007), and Elnikety et al., (2009) for measuring database performance on small-scale servers. Stability of measured results was addressed by employing the TPC-W benchmark, which embodies well-known workload characteristics (Menascé & Almeida, 2001). Predictability of the measurements in the experiments conducted was addressed using an experimental design that included replication, thus allowing the calculation of experimental error (Jain, 1991).

Internal validity in laboratory experiments describes the confidence with which a researcher may state that the independent variable A causes changes in the dependent variables B, and C (Sekaran, 2003). Validity in this context refers to the degree to which

the experimental measurements establish or refute relationships between variables (Salkind, 2006). A $2^k r$ experimental design was chosen for the experiments conducted in order to allow the calculation of experimental error. Thus, results reported can be show to be statistically significant with a reasonably high degree of validity.

External validity for the experimental work conducted was established by evaluating the FACE model against a real-world data set. According to Sekaran (2003), cause-and-effect relationships established in the laboratory should be evaluated in a field setting to establish external validity. However as noted by Keeton and Patterson (2000), availability of large-scale systems for field-testing is not feasible for most research projects. Instead, a natural model constructed using components of a real-world system provides a practical alternative to artificial models like the TPC-W benchmark (Menascé & Almeida, 2001). The wildlife habitat capability-suitability is viewed by the author as a good candidate for use in establishing the generalizability of the FACE model as it constitutes a natural model that is both simple and relatively compact.

Summary

The demand for secure Web-based computing provides the motivation for wider adoption of fine-grained access control (Zhu and Lü, 2007). The contribution of the author's study is a generic model, termed the FACE model. The FACE model quantifies the performance overhead of four common implementations of fine-grained access control.

The goal in undertaking the author's study was to extend and consolidate work on the performance aspects of fine-grained access control previously reported in the

literature. According to Kabra et al. (2006), understanding the performance implications of fine-grained access control is essential to use of the technology. The author employed experiment as the methodology for the study. Sharp and Howard (as cited in Dawson, 2000) describe the nature of this work as applied research.

Chapter 4

Results

Analysis

In the author's study, four experiments were conducted following the methodology described in Chapter 3. The first three experiments consisted of 150 trials, each with three repetitions. The first 75 trials evaluated a control and four different fine-grained access control mechanisms using read-only queries based upon the TPC-W benchmark. The second 75 trials evaluated update transactions from the TPC-W benchmark. The fourth experiment consisted of 25 trials, each with three repetitions, employing a control and four different fine-grained access control mechanisms. In the fourth experiment, read-only queries against a large, real-world data set were evaluated. The queries employed for benchmarking in the fourth experiment were obtained from an online wildlife habitat capability/suitability application. The wildlife application, now defunct, did not contain any INSERT or UPDATE transactions.

Validating the Experimental Configuration

Zhu and Lü (2007) indicated that a TPC-W workload generated by four thousand EBS's could be sustained on a small-scale server. To verify server capacity of the author's experimental configuration, a complex TPC-W query was run against a dedicated server hosting a PostgreSQL database populated with 10 million rows in the ITEM table. Workload was managed using Jmeter to emulate 4000 EBS's, which was the

largest TPC-W workload expected to be employed in the author's study. Server performance data was collected using the Linux system activity reporter (SAR) utility. The SAR utility captures four statistics – percentage of time waiting on system processes (System), percentage of time waiting on user processes (User), percentage of time waiting on disk devices (Wait), and percentage of time the CPU is idle (Idle). As depicted in Figure 15, CPU time expended executing System (green) and User (red) processes never exceeded 60% while executing the test query, demonstrating that the small-scale server employed by the author provided sufficient capacity for the TPC-W workloads to be evaluated in the study.

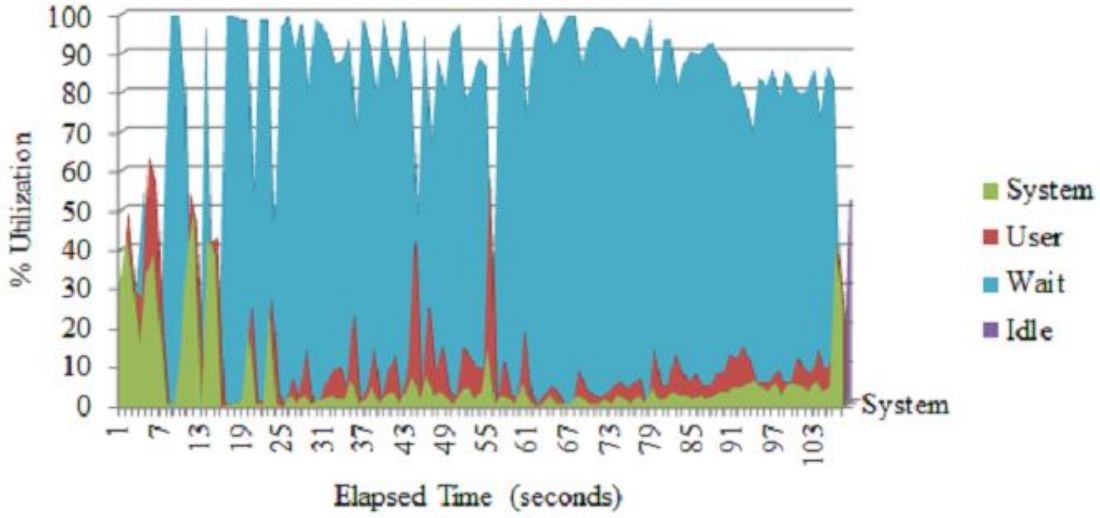

Figure 15. Server Load while Executing a TPC-W Workload with 4000 EBS

Bitton et al. (1983), and Keeton and Patterson (2000) observed that when benchmarking SQL queries, best results are obtained when the SQL queries generate disk access requests rather than retrieving data already resident in the database buffer cache. Server activity statistics depicted in Figure 15 illustrates that the greatest proportion of server activity during execution of the author's test query was spent waiting

for data to be read from disk. This test confirmed that for an ITEM table containing 10 million rows, the TPC-W query evaluated was not able to satisfy the query using data residing in the buffer cache and was therefore required to fetch data from disk.

Kalibera and Tuma (2006) indicated that fluctuations caused by data caching could distort benchmark performance measures. In order to minimize the influences of data caching on benchmark results, the database buffers and the operating system file cache were flushed prior to running each repetition of the author's experiments.

Figure 16 depicts the results when the same TPC-W query employed for evaluating server capacity was executed both with and without flushing the database buffers and operating system file caches between repetitions of the same experiment. Fluctuations due to caching were most pronounced when executing the selected query against an ITEM table containing one hundred thousand rows and to a lesser degree, against an ITEM table containing one million rows. As predicted by Bitton et al. (1983), the effects of caching are negligible when the query is executed against a table that is significantly larger than the capacity of the database buffer cache. When the selected TPC-W query was executed against an ITEM table containing 10 million rows, flushing the buffer cache between repetitions of the same experiment produced results similar to experiments performed without flushing the buffer cache.

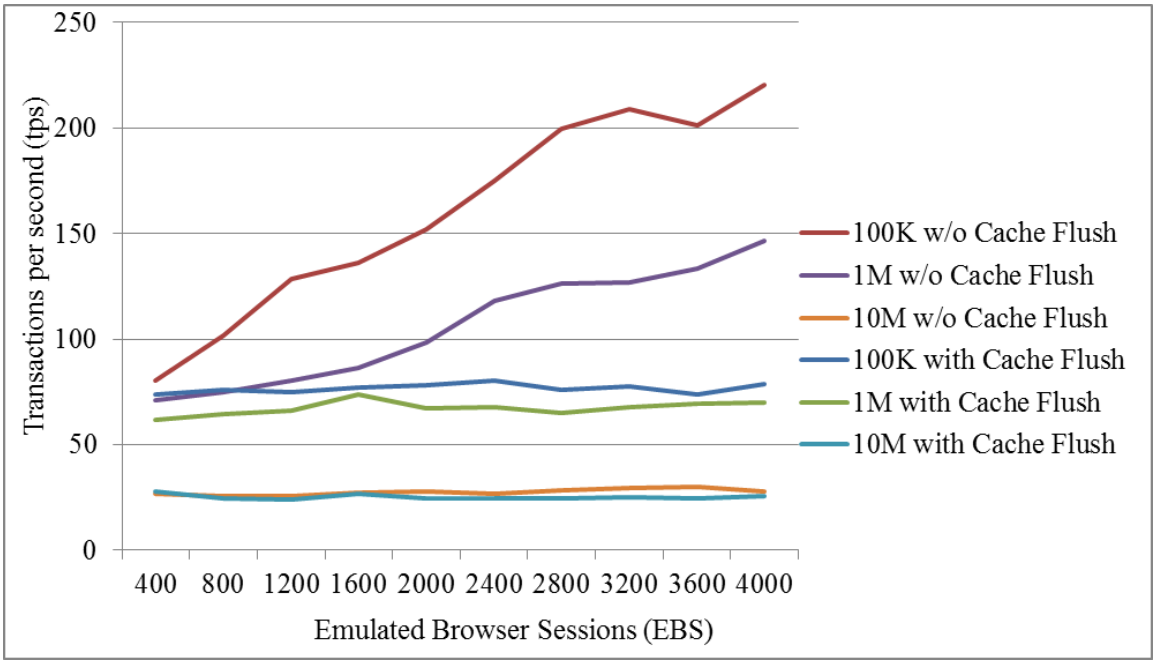

Figure 16. Effect of Caching on TPC-W Query Throughput

Findings

The results obtained by the author during the experimentation phase of the study were recorded using the sign table methodology as described by Jain (1991). Detailed results are presented in tabular form in Appendix M through Appendix P. The sign table methodology also provides a simple means for calculating variation attributed to each experimental variable. The author extended the standard sign table format to include calculation of variation due to each experimental variable and calculation of standard error for each set of measurements. Microsoft Excel was used to automate the calculations for each sign table. The formulas described in Equations 1.1 through 1.10 in Chapter 3 are used as the basis of the calculations in the Excel spreadsheets used to generate the sign tables. Calculations of mean values are based upon the geometric mean in cases where the underlying data consists of normalized values. As discussed by Dixit

(1992), aggregated benchmark results based upon elapsed time of execution are to be averaged using the geometric mean rather than the arithmetic mean.

The first experiment evaluated the performance of read-only (query) transactions and read-write (update) transactions against selected queries from the TPC-W benchmark populated with one hundred thousand items. There were three repetitions of each trial within the experiment for each of the access control mechanisms evaluated. Query durations reported are based upon the geometric mean of the three repetitions. The database and O/S buffers were flushed prior to the execution of each benchmark. Confidence intervals for the experiment are depicted in Table 7. With 95% confidence, the margin of error is less than ±5%. Results for this experiment are summarized in Figure 17, Figure 18, Figure 19, and Figure 20.

Table 7. Confidence intervals for TPC-W benchmarks - 100,000 Items

Items	Fine-Grained Access Control Mechanism	s_e	s_{qi}	90%	95%
100,000	Authorization Views	75.97	15.51	±20.73	±27.08
100,000	Hippocratic Database	62.36	12.73	±17.02	±22.23
100,000	Label Based Access Control	65.25	13.32	±17.81	±23.26
100,000	Transparent Query Rewrite	57.49	11.74	±15.69	±20.49

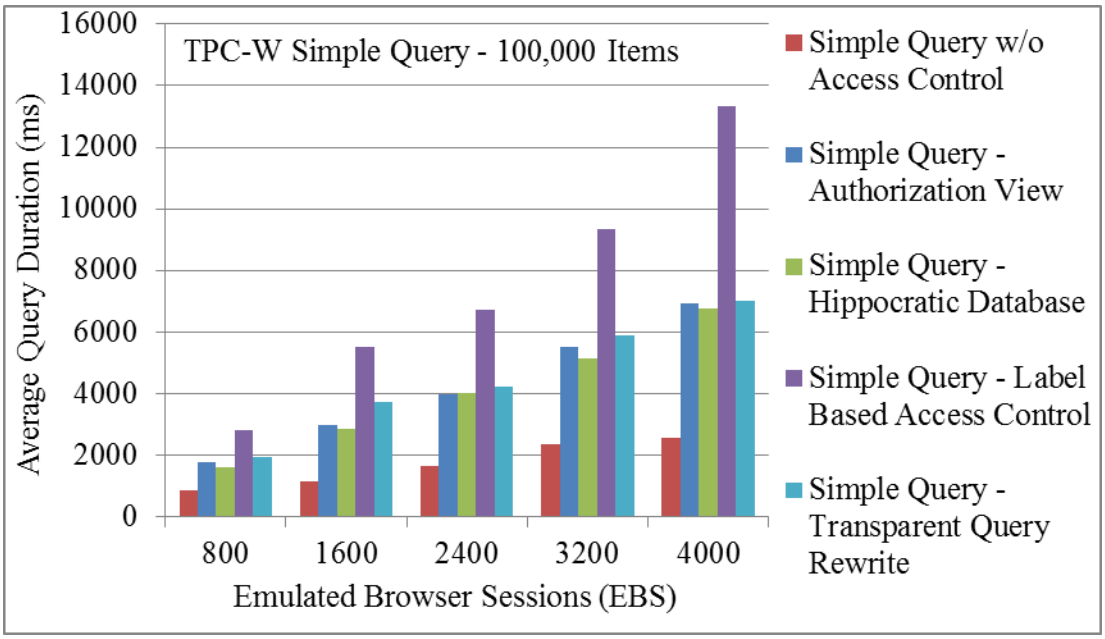

Figure 17. Average query duration for TPC-W simple queries - 100,000 Items

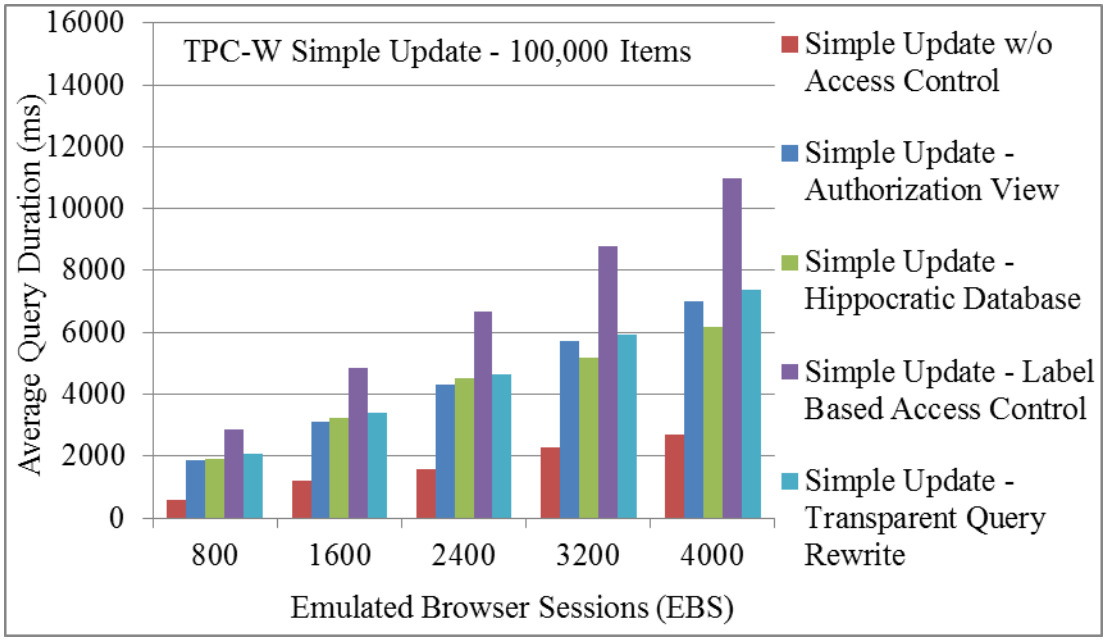

Figure 18. Average query duration for TPC-W simple updates - 100,000 Items

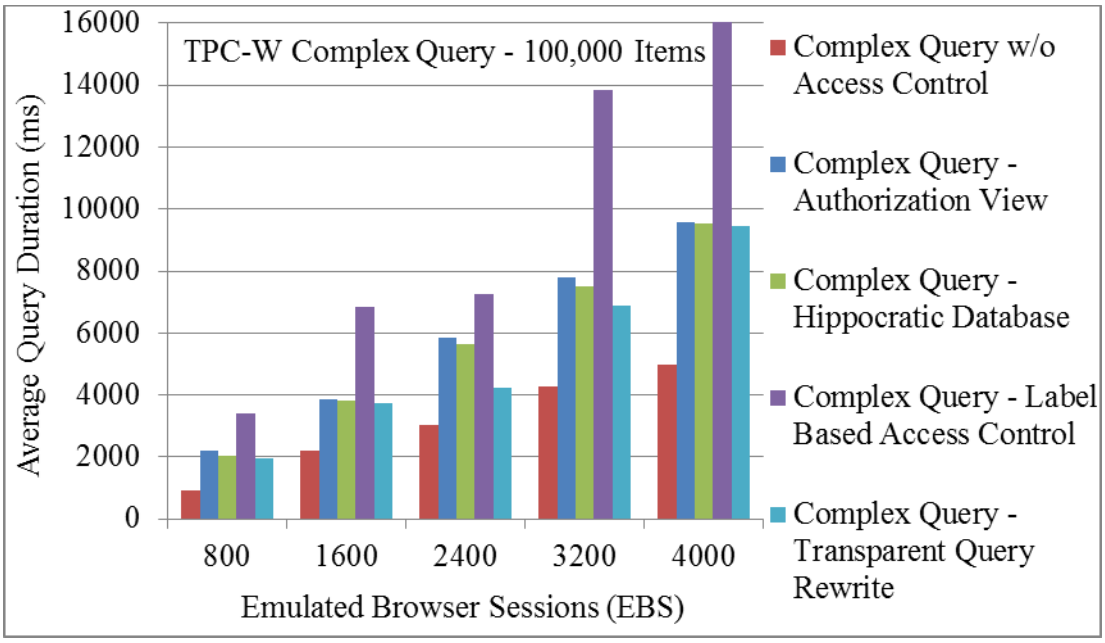

Figure 19. Average query duration for TPC-W complex queries - 100,000 Items

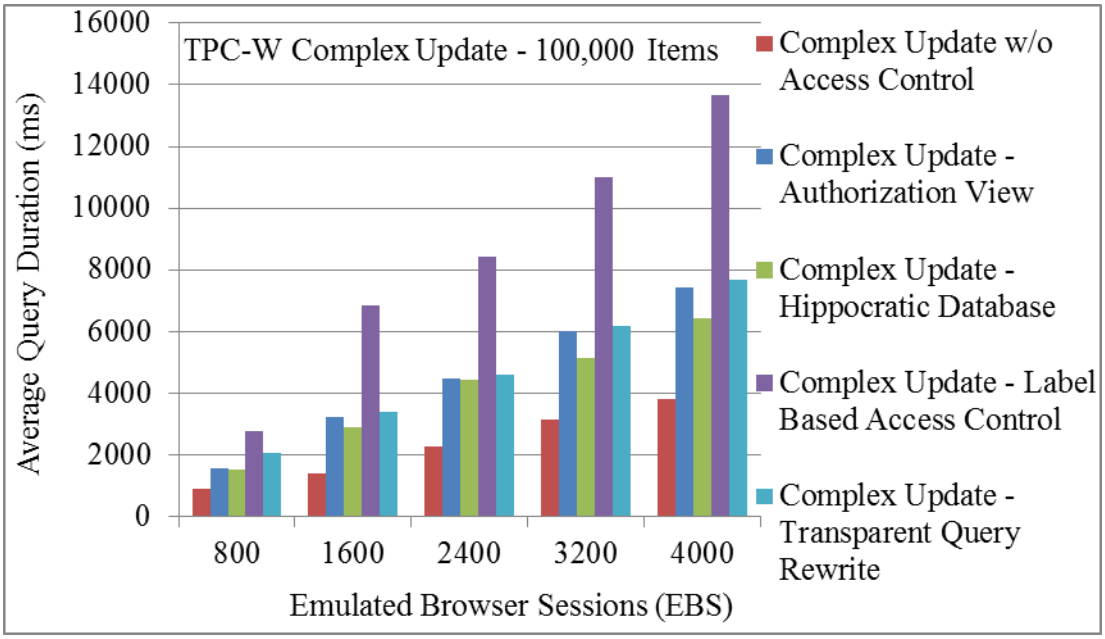

Figure 20. Average query duration for TPC-W complex updates - 100,000 Items

The second experiment evaluated the performance of read-only (query) transactions and read-write (update) transactions against selected queries from the TPC-W benchmark populated with one million items. There were three repetitions of each trial within the experiment for each of the four access control mechanisms evaluated. Query durations reported are based upon the average of the three repetitions. The database and O/S buffers were flushed prior to the execution of each benchmark. Confidence intervals for the experiment are depicted in Table 8. With 95% confidence, the margin of error is less than ±5%. Results for this experiment are summarized in Figure 21, Figure 22, Figure 23, and Figure 24.

Table 8. Confidence intervals for TPC-W benchmarks - One Million Items

Items	Fine-Grained Access Control Mechanism	s_e	s_{qi}	90%	95%
1M	Authorization Views	68.	15.51	±20.73	±27.08
1M	Hippocratic Database	60.67	12.38	±16.56	±21.62
1M	Label Based Access Control	64.66	13.20	±17.65	±23.04
1M	Transparent Query Rewrite	64.97	13.26	±17.73	±23.16

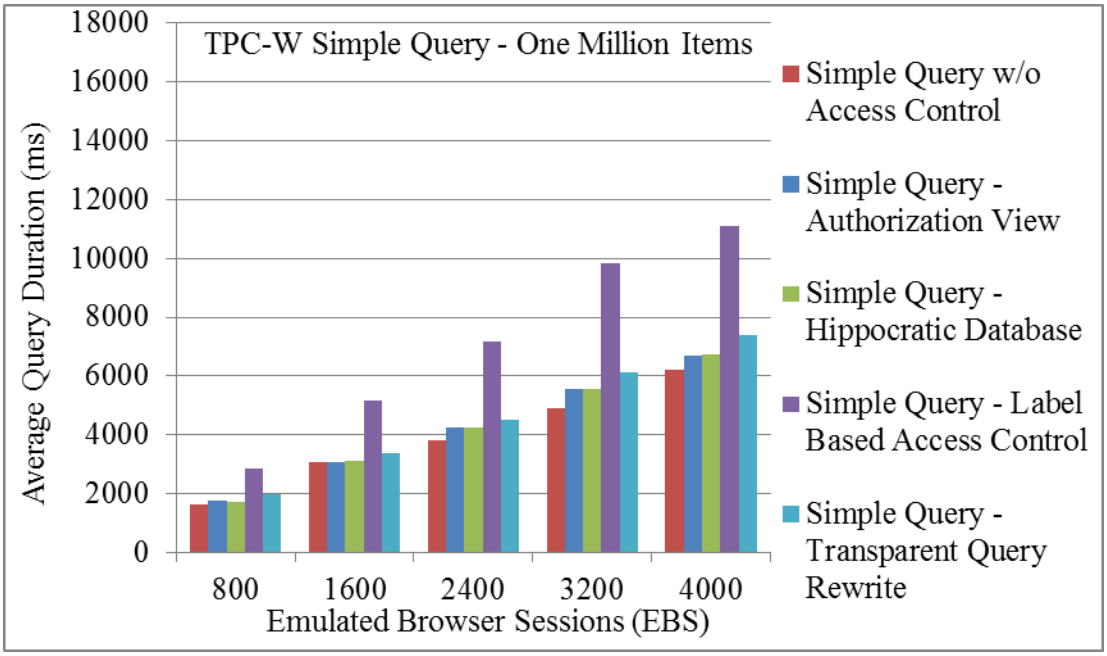

Figure 21. Average query duration for TPC-W simple queries – One Million Items

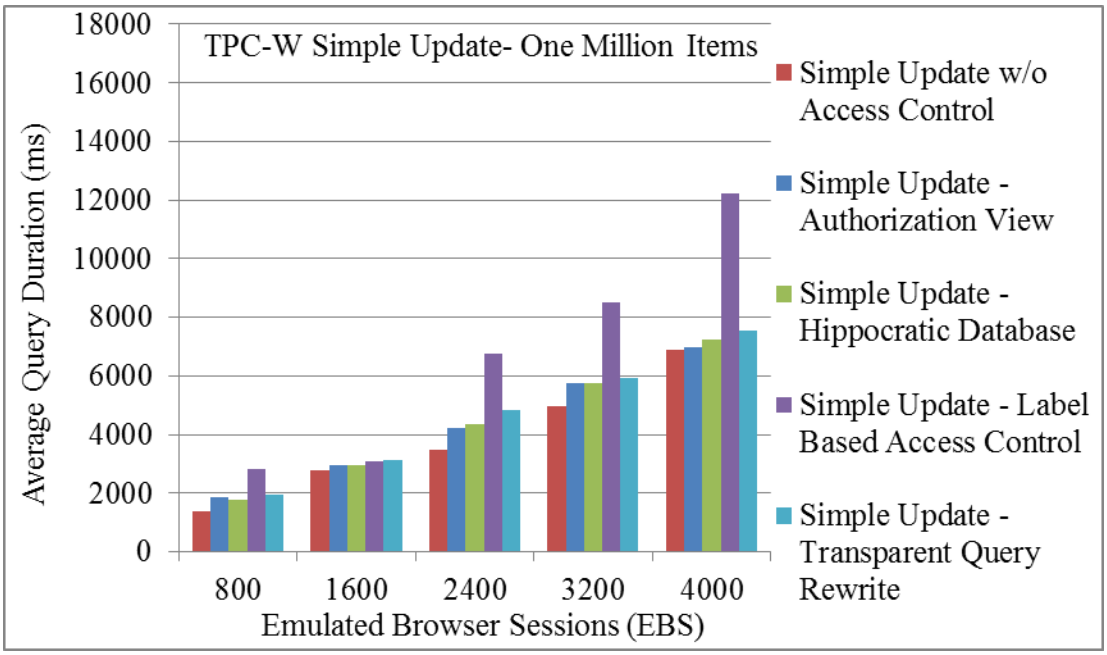

Figure 22. Average query duration for TPC-W simple updates – One Million Items

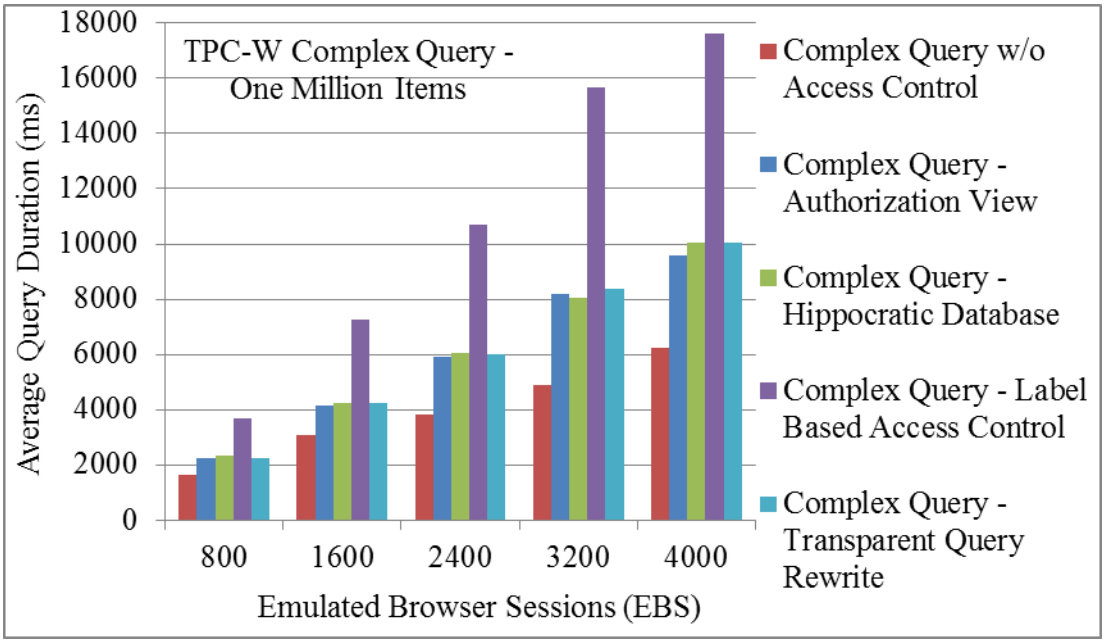

Figure 23. Average query duration for TPC-W complex queries – One Million Items

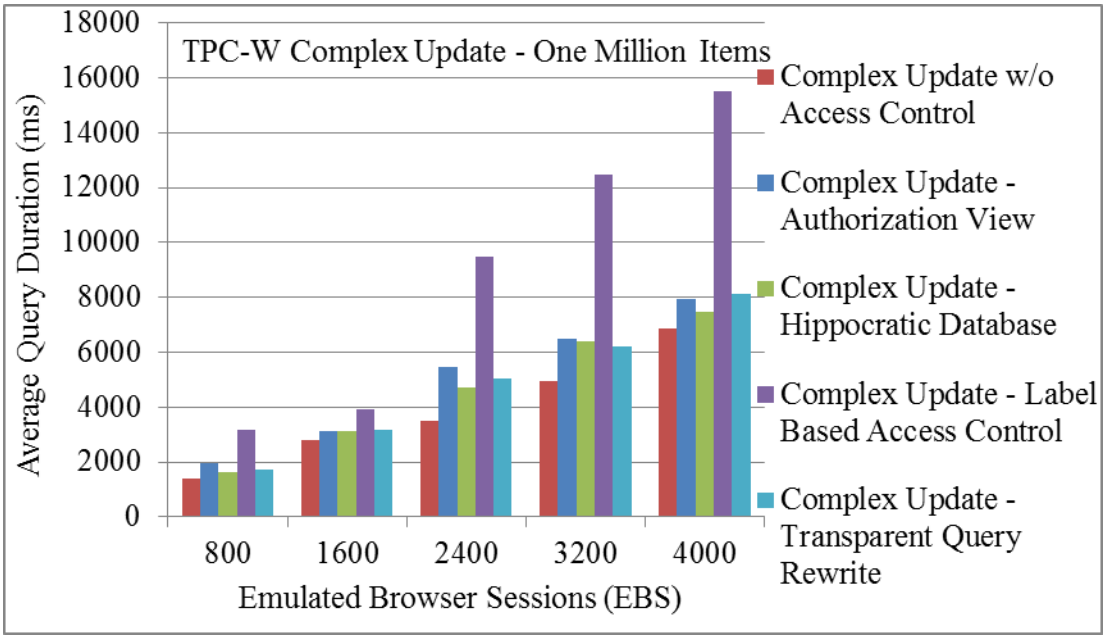

Figure 24. Average query duration for TPC-W complex updates – One Million Items

The third experiment evaluated the performance of read-only (query) transactions and read-write (update) transactions against selected queries from the TPC-W benchmark populated with 10 million items. There were three repetitions of each trial within the experiment for each of the four access control mechanisms evaluated. Query durations reported are based upon the average of the three repetitions. The database and O/S buffers were flushed prior to the execution of each benchmark. Confidence intervals for the experiment are depicted in Table 9. With 95% confidence, the margin of error is less than ±5%. Results for this experiment are summarized in Figure 25, Figure 26, Figure 27, and Figure 28.

Table 9. Confidence intervals for TPC-W benchmarks – 10 Million Items

Items	Fine-Grained Access Control Mechanism	s_e	s_{qi}	90%	95%
10M	Authorization Views	75.97	15.51	±20.73	±27.08
10M	Hippocratic Database	50.02	10.21	±13.65	±17.83
10M	Label Based Access Control	55.93	11.42	±15.26	±19.93
10M	Transparent Query Rewrite	52.76	10.77	±14.40	±18.80

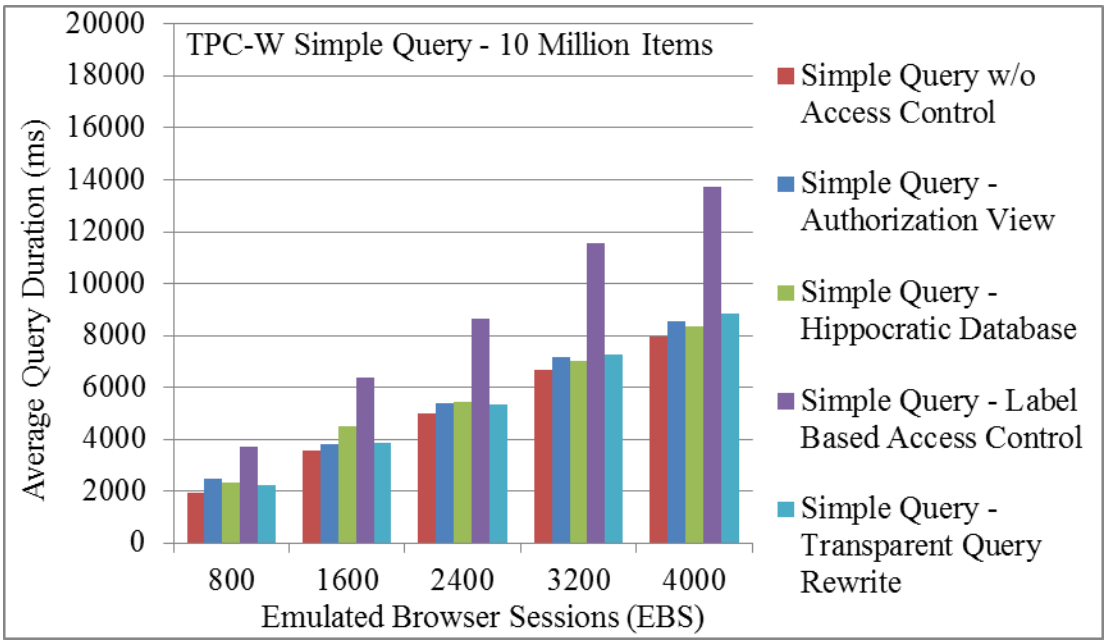

Figure 25. Average query duration for TPC-W simple queries – 10 Million Items

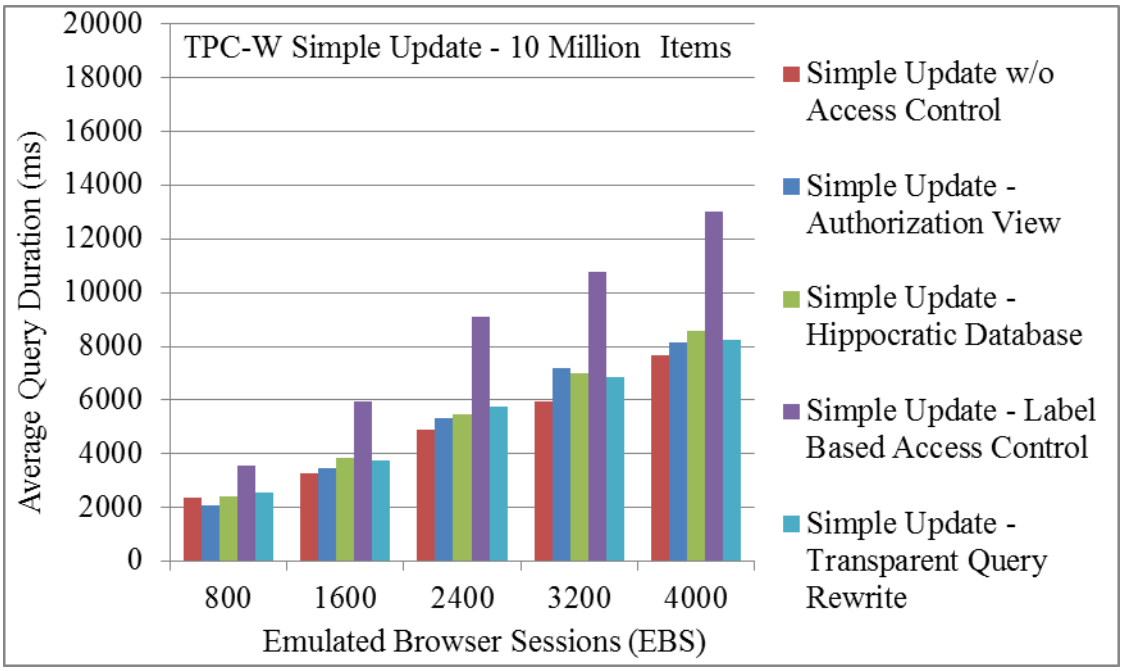

Figure 26. Average query duration for TPC-W simple updates – 10 Million Items

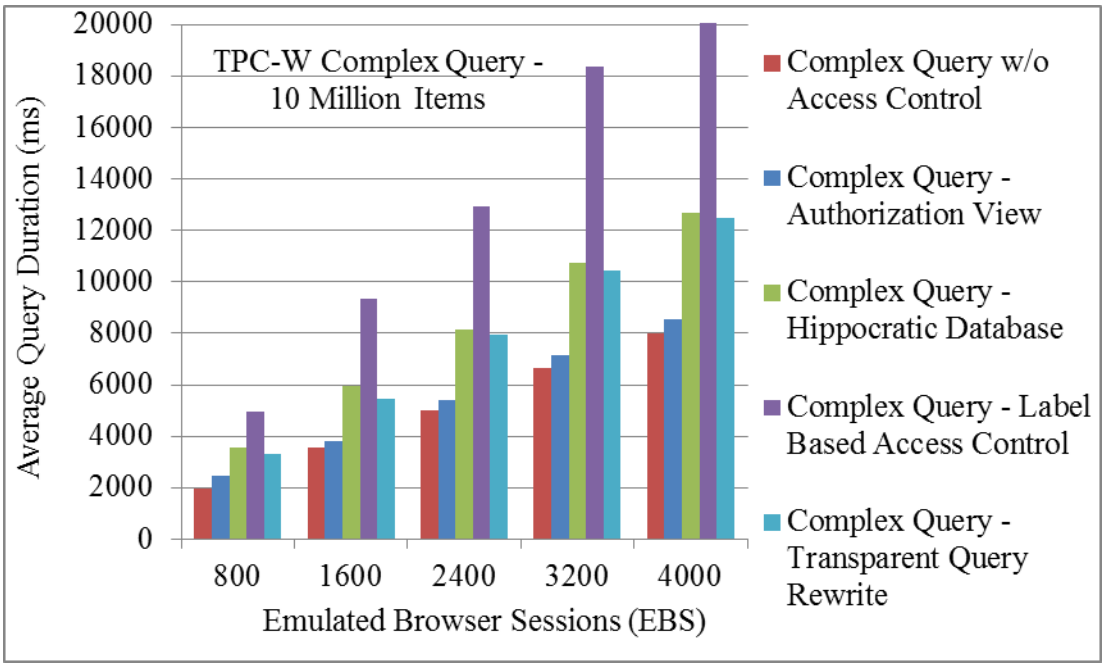

Figure 27. Average query duration for TPC-W complex queries – 10 Million Items

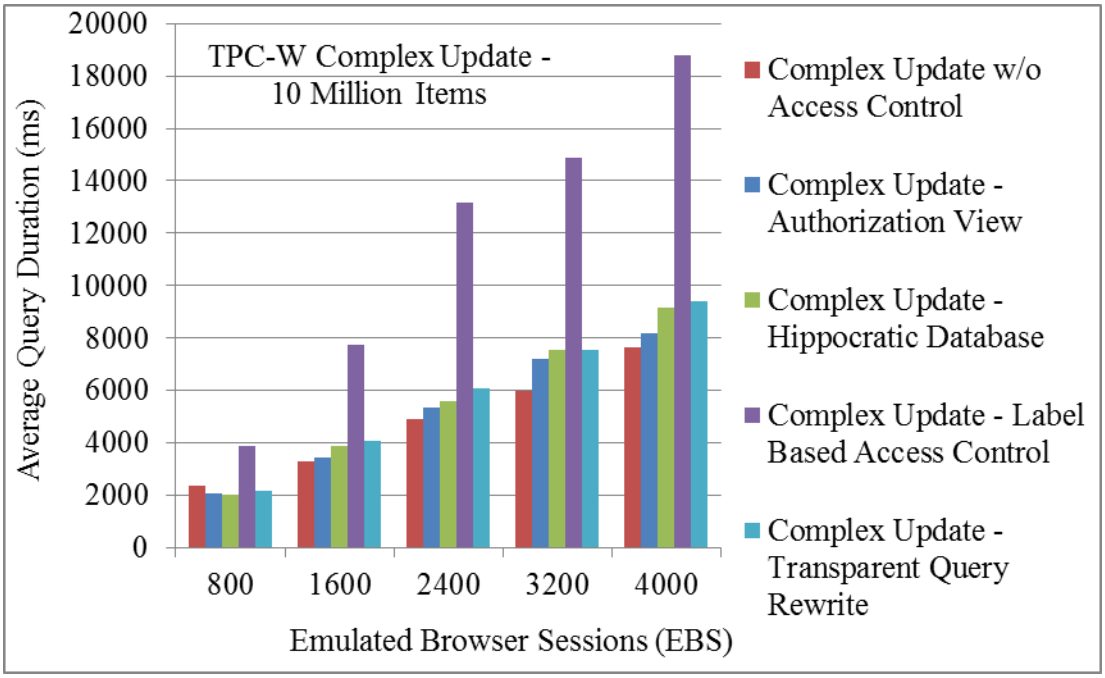

Figure 28. Average query duration for TPC-W complex updates – 10 Million Items

The fourth experiment evaluated the performance of read-only (query) transactions against data from the real-world wildlife application. There were three repetitions of each trial within the experiment for each of the four access control mechanisms that were evaluated. Query durations reported are based upon the average of the three repetitions. The database and O/S buffers were flushed prior to the execution of each benchmark. Confidence intervals for the experiment are depicted in Table 10. With 95% confidence, the margin of error is less than ±5%. The results for this experiment are summarized in Figure 29 and Figure 30. The difference in query duration between the simple and complex queries against the wildlife data is approximately an order of magnitude. Therefore, the y-axis of the histograms depicted in Figure 29 and Figure 30 have been formatted using a logarithmic scale to allow easier comparison between the two graphs.

Table 10. Confidence intervals for Wildlife benchmarks

Items	Fine-Grained Access Control Mechanism	s_e	s_{qi}	90%	95%
Wildlife	Authorization Views	224.65	64.85	±90.598	±120.62
Wildlife	Hippocratic Database	169.96	227.24	±317.46	±422.67
Wildlife	Label Based Access Control	153.96	205.85	±287.57	±382.87
Wildlife	Transparent Query Rewrite	121.03	161.82	±226.06	±300.99

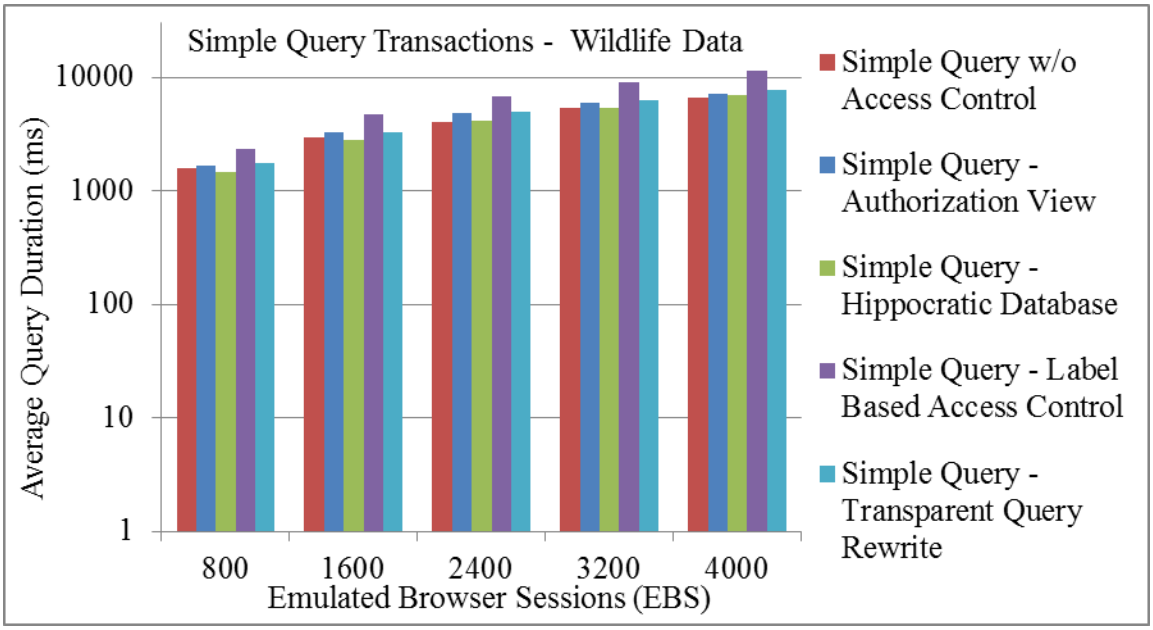

Figure 29. Average query duration for Wildlife simple queries (log scale)

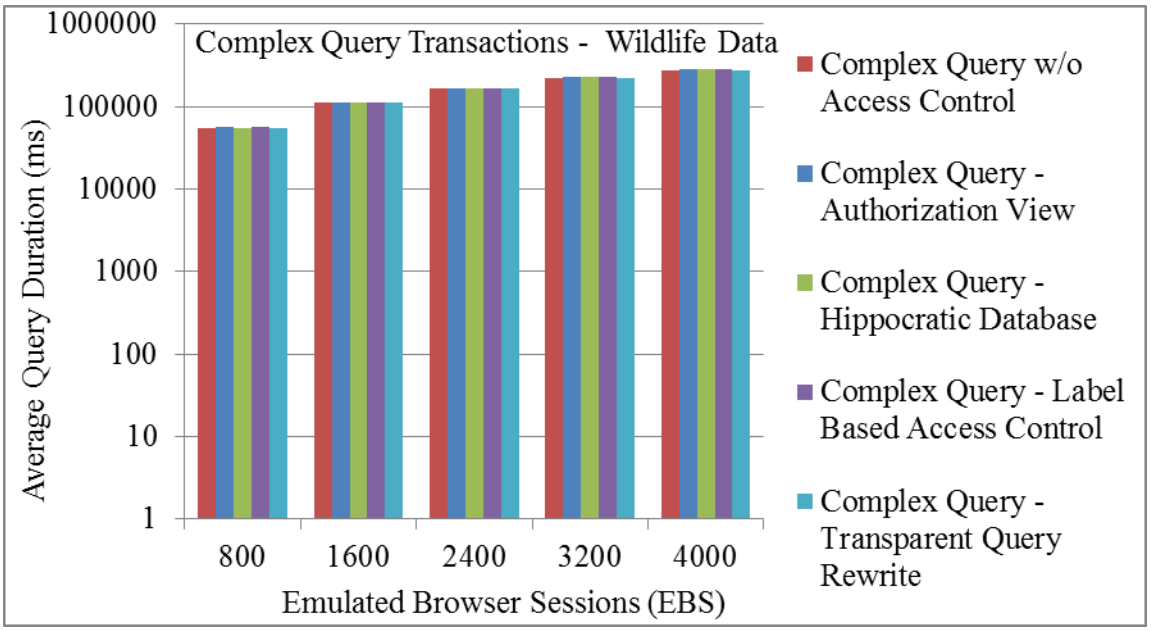

Figure 30. Average query duration for Wildlife complex queries (log scale)

Scalability

One of the goals of the author's study was to determine scalability of the four fine-grained access control mechanisms evaluated. Figure 31 depicts the average response time for the control (i.e., no fine-grained access control) and four fine-grained access control mechanisms using a complex query from the TPC-W benchmark suite. The y-axis of the graph depicts the average response time in milliseconds (ms) and the x-axis depicts the number of EBS's (i.e., workload). Figure 32 provides a similar line-graph for the wildlife data where response times for complex queries are measured for the control and four fine-grained access control mechanisms. Note that the axes for Figure 31 and Figure 32 have the same scale. Following the methodology of Elnikety et al. (2009) for visual analysis of response time versus load for TPC-W complex query (i.e., read-

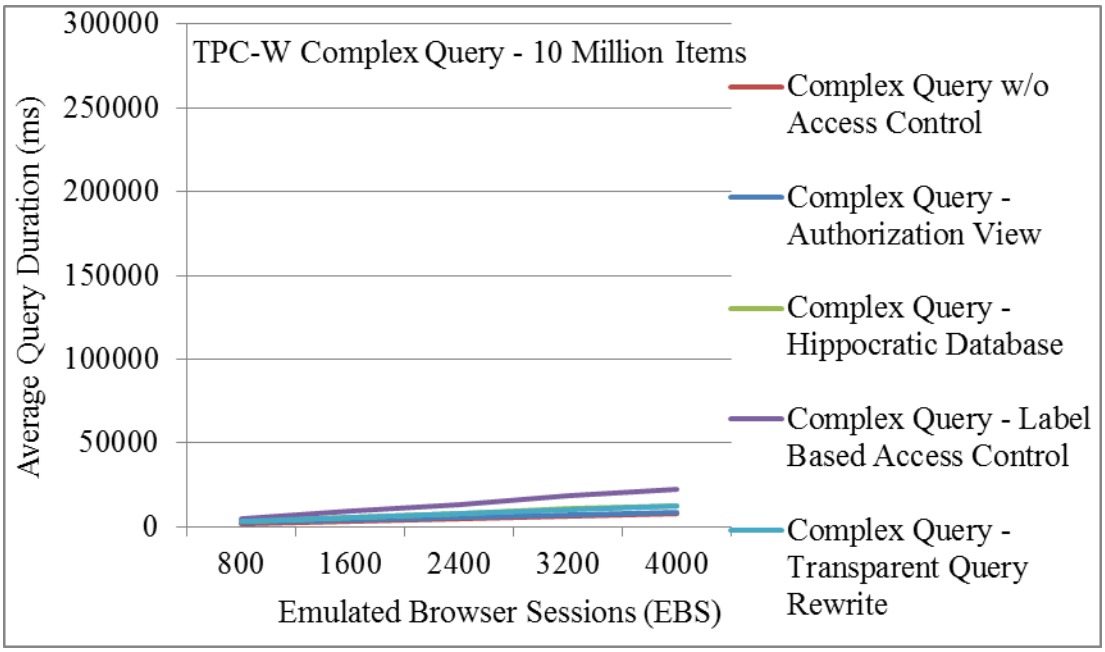

Figure 31. TPC-W response time for complex queries – 10 Million Items

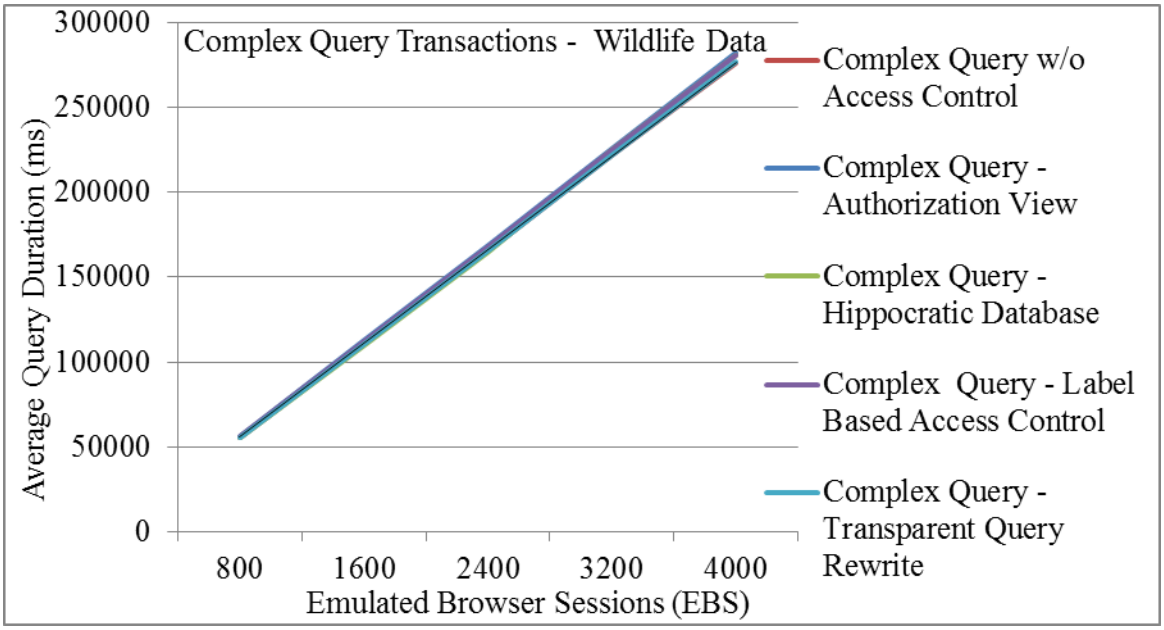

Figure 32. Wildlife response times for complex queries

only) transactions, it can be observed in Figure 31 that response time curves are nearly flat, which is indicative of excellent scalability for transaction processing that contains little or no update transactions. These findings confirm the statement by Menascé (2002) that the TPC-W benchmark exhibits excellent scalability. In contrast, the response time curves depicted in Figure 32 for the wildlife data, while similar for the control and each of the fine-grained access control mechanisms evaluated, indicate that scalability for the wildlife read-only transactions is linear. According to Gunther (2004), the linear response time curves observed in the wildlife benchmarks indicate that the workload is scaling as best as it can with the limited computing resources provided by the small-scale processor.

FACE Model

The sign table methodology provides a simple means for calculating variation attributed to each experimental variable. Table 11 summarizes the percentage of variation for the TPC-W complex query scaled to an ITEM table of 10 million rows. Table 11 also summarizes the percentage of variation for a complex query against the wildlife data.

Table 11. Overhead imposed by fine-grained access control

Items	Fine-Grained Access Control Mechanism	Fine-Grained Access Control	Query Complexity	SQL Operation
10 Million	Authorization Views	3%	31%	33%
10 Million	Hippocratic Database	6%	27%	33%
10 Million	Label Based Access Control	69%	18%	5%
10 Million	Transparent Query Rewrite	5%	27%	27%
Wildlife	Authorization Views	1%	99%	N/A
Wildlife	Hippocratic Database	7%	93%	N/A
Wildlife	Label Based Access Control	7%	93%	N/A
Wildlife	Transparent Query Rewrite	7%	93%	N/A

Figure 33 depicts a throughput graph for TPC-W complex queries scaled to 10 million items. Figure 34 depicts a response time graph for TPC-W complex queries scaled to 10 million items. Figure 35 depicts a throughput graph for wildlife complex queries. Figure 36 depicts a response time graph for wildlife complex queries.

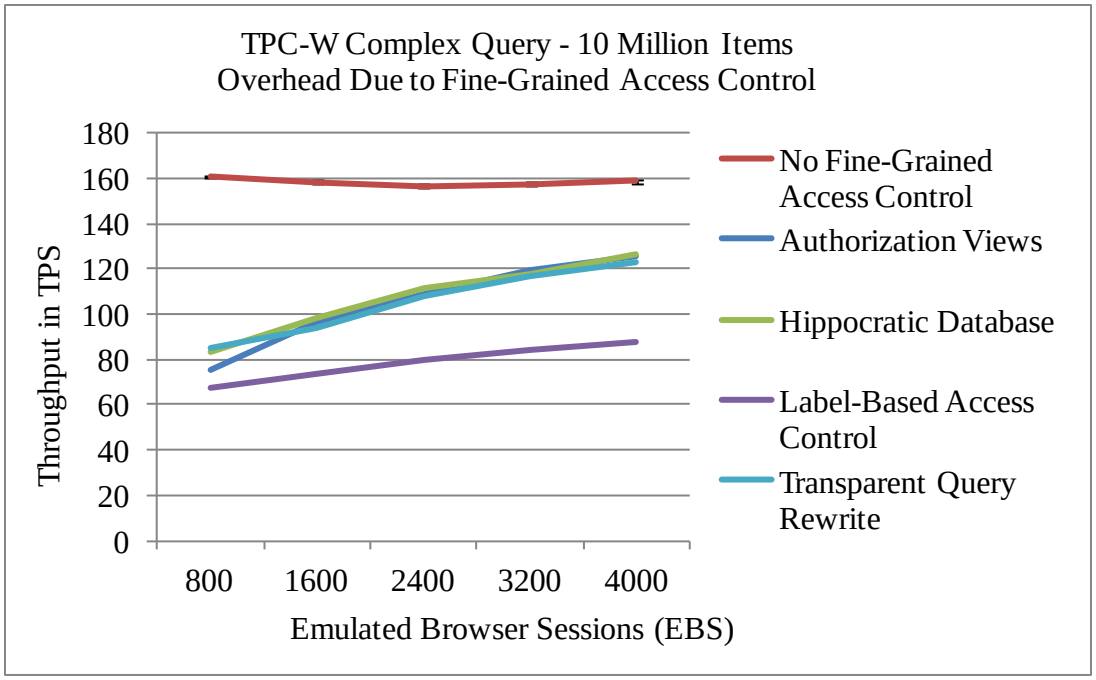

Figure 33. Throughput for TPC-W complex queries - 10 Million Items

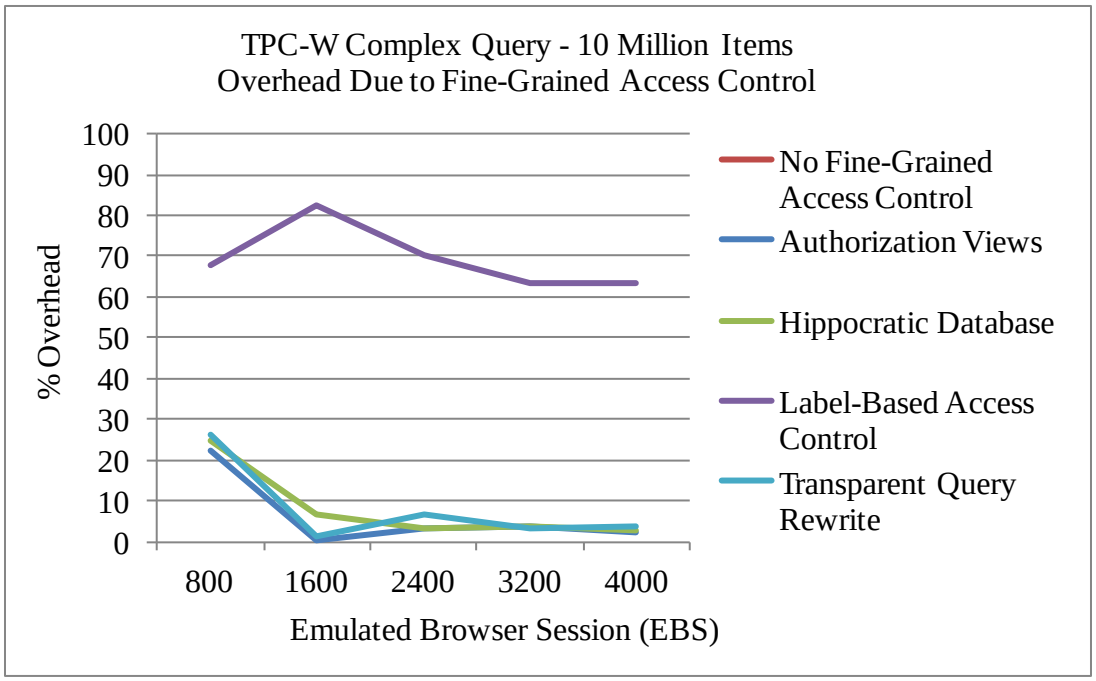

Figure 34. Overhead for TPC-W complex queries – 10 Million Items

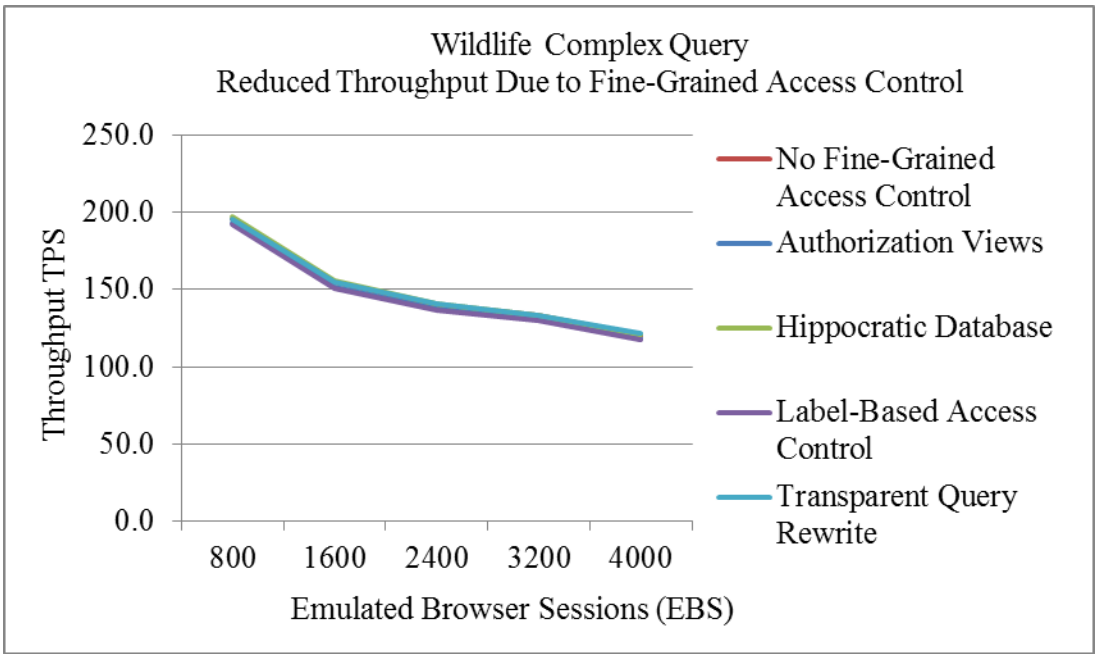

Figure 35. Throughput for Wildlife complex queries

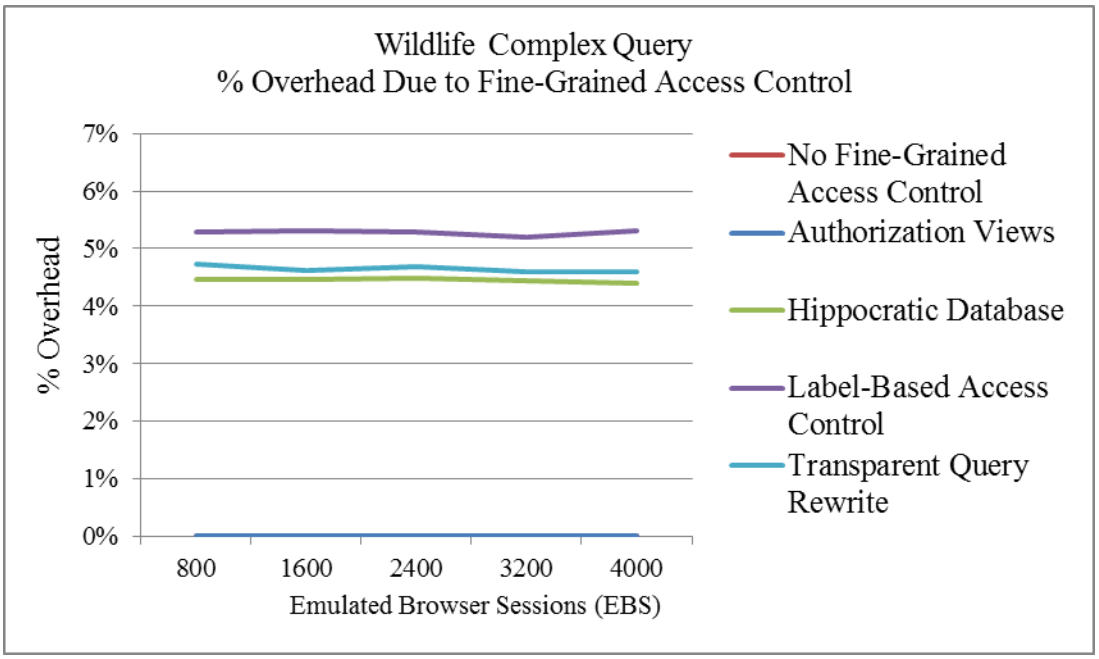

Figure 36. Overhead for Wildlife complex queries

The slopes of the throughput line plots (Figure 33) for the TPC-W complex queries are relatively flat for workloads with more than 2400 EBS's, indicating that the workload scales linearly. The corresponding response time graph (Figure 34) indicates that the percentage of processing overhead for the TPC-W complex queries due to fine-grained access control is low (<6%) except for LBAC, where the overhead is nearly 70%. The slopes of throughput line plots (Figure 35) for the wildlife complex queries depict a gradual negative slope indicating that the small-scale processor has reached its maximum throughput (Menascé and Almeida, 2001). The corresponding response time graph (Figure 34) for the wildlife complex queries indicates that the percentage of processing overhead due to fine-grained access control remains constant (<7%) for all of the fine-grained access control mechanisms evaluated.

Summary of Results

In the author's study, queries from the TPC-W benchmark and queries from a real-world wildlife habitat capability/suitability application were used to quantify the overhead imposed by the use of fine-grained access control. Four fine-grained access control mechanisms were evaluated – authorizations views, the Hippocratic database, LBAC, and transparent query rewrite. Benchmark results were summarized using sign tables and graphed using histograms to provide a simple visual comparison of the mechanisms evaluated. In terms of average query duration, a simple database query against the wildlife data was demonstrated to be roughly equivalent to a complex query against the TPC-W data scaled to 10 million items. Complex queries against the wildlife data had an average query duration that was 10 times greater than the wildlife simple

queries. Based upon measurements of average query duration, there were approximately three orders of magnitude between the simplest TPC-W query evaluated and the most complex query executed against the wildlife data. Thus, the workload employed in the author's study was demonstrated to provide good scalability.

The test results from the first experiment revealed that for simple read-only transactions executed against a PostgreSQL database scaled to one hundred thousand TPC-W items, the overhead imposed by the use of authorization views, the Hippocratic database, and transparent query rewrite were roughly equivalent. In contrast, the use of LBAC for simple read-only transactions was observed to impose performance costs that were nearly 50% greater than the overhead imposed by the three other fine-grained access control mechanisms. In the case of simple read-write transactions executed against a PostgreSQL database scaled to one hundred thousand TPC-W items, the overhead imposed by the use of authorization views, the Hippocratic database, and transparent query rewrite were similar to the performance overhead observed for simple read-only transactions. However, the total overhead per transaction was marginally higher. As was the case for simple read-only transactions, the use of LBAC was observed to impose performance costs that were approximately 50% greater than the overhead imposed by the other three fine-grained access control mechanisms.

Test results from the first experiment revealed that for complex read-only transactions executed against a PostgreSQL database scaled to one hundred thousand TPC-W items, the overhead imposed by the use of authorization views, the Hippocratic database, and transparent query rewrite were roughly equivalent. Complex read-write transactions executed against the same database exhibited similar characteristics although

the overhead per transaction was lower than for read-only transactions, which is the opposite behavior observed for simple transactions and updates. However as was the case with complex read-only and read-write transactions, the use of LBAC was observed to imposed performance costs nearly 50% greater than the overhead imposed by the other three fine-grained access control mechanisms.

Test results from the second experiment revealed that for simple read-only transactions and simple read-write transactions executed against a PostgreSQL database scaled to one million TPC-W items, the overhead imposed by the use of authorization views, the Hippocratic database, and transparent query rewrite were roughly equivalent. The use of LBAC for simple read-only transactions and simple read-write transactions was observed to impose performance costs that were nearly 50% greater than the overhead imposed by the other three fine-grained access control mechanisms. In the case of complex read-only transactions and complex read-write transactions executed against a PostgreSQL database scaled to one million TPC-W items, performance costs associated with the use of fine-grained access control were similar to simple read-only and simple read-write transactions. However, the performance impacts were lower for the read-write transactions than for read-only transactions.

Test results for the third experiment revealed that for simple read-only transactions and simple read-write transactions executed against a PostgreSQL database scaled to 10 million TPC-W items, the overhead imposed by the use of authorization views, the Hippocratic database, and transparent query rewrite were roughly equivalent. The use of LBAC for simple read-only transactions and simple read-write transactions was observed to impose performance costs that were nearly 50% greater than the

overhead imposed by the three other fine-grained access control mechanisms. In the case of complex read-only transactions and complex read-write transactions executed against a PostgreSQL database scaled to 10 million TPC-W items, performance costs associated with the use of fine-grained access control were similar to simple read-only and simple read-write transactions. However, the performance impacts were lower for the read-write transactions than for read-only transactions.

Test results for the fourth experiment revealed that for simple read-only transactions executed against a PostgreSQL database containing data from the real-world wildlife application, the overhead imposed by the use of authorization views, the Hippocratic database, and transparent query rewrite were equivalent. The use of LBAC imposed only slightly greater performance costs than for the other fine-grained access control mechanisms. In the case of complex read-only transactions executed against a PostgreSQL database containing data from the real-world wildlife application, performance costs associated with the use of fine-grained access control were similar to simple read-only transactions. Significantly, there were no discernible differences in the performance overhead associated with authorization views, the Hippocratic database, LBAC, and transparent query rewrite.

One of the goals of the study was to quantify the scalability of the four selected fine-grained access control mechanisms. Scalability was established by comparing average query duration versus workload as described by Elnikety et al. (2009). The query duration versus workload line graphs for the author's TPC-W benchmarks were very similar to the research findings reported by Elnikety et al. for TPC-W benchmark workloads. The query duration versus workload graphs for the wildlife data benchmarked

by the author were linear, indicating that query processing that includes fine-grained access control mechanisms, are scalable within the processing capacity of the small-scale server used for benchmarking.

The FACE model employs simple, line graphs to represent the scalability and overhead imposed by the four fine-grained access control mechanisms evaluated. The results presented indicate that the overhead for complex queries against large data sets is minimal, typically 7% or less. The slope of the response time curves indicate that under higher workload, specifically 2400 EBS's or greater, that the fine-grained access control mechanisms are scalable. It should be noted however that in the sign tables presented in Appendix M and Appendix N for TPC-W benchmarks scaled to one hundred thousand items and one million items respectively, indicate that the proportion of overhead due to the use of fine-grained access control is significantly higher for data sets containing a modest number of rows.

Chapter 5

Conclusions, Implications, Recommendations and Summary

Conclusions

The FACE model provides a simple approach to quantifying the scalability and performance of typical fine-grained access control mechanisms used with relational database management systems. The literature contains references to approaches that quantify the performance and scalability of individual fine-grained access control mechanisms used with relational database management systems. However, quantification of the performance overhead for multiple fine-grained access control mechanisms against the same data has not been previously described.

A novel aspect of the author's study was the evaluation of performance and scalability associated with read-write transactions where fine-grained access control was implemented. Most of the current literature describing fine-grained access control focuses solely upon read-only transactions. In the author's study, it was demonstrated that in the case of SQL UPDATE transactions, where fine-grained access control was implemented, scalability and performance overhead were equivalent to read-only transactions.

The existing research studies suggest that some fine-grained access control mechanisms are not scalable. For example, Bertino et al. (2006) suggest that authorization views are not scalable due to the associated overhead with managing a large number of views. However, this is quite different from scalability of authorization views from the standpoint of performance overhead. As demonstrated in Figure 31 and Figure

32, authorization views, the Hippocratic Database, LBAC, and transparent query rewrite are all scalable mechanisms from the standpoint of performance overhead versus workload. Contrary to widely held beliefs in the database community that the use of fine-grained access control can impose onerous performance overhead, the simple visual model provided by FACE demonstrates that the performance impacts tend to be minimal, representing an acceptable compromise between enhanced security and reduced performance.

Implications

Based upon the results of this study, a number of current approaches to implementing fine-grained access control for relational database management systems are deemed eminently practical for use in securing large Web-based applications. All of the fine-grained access control mechanisms evaluated in the author's study, with the exception of LBAC, may be deployed without concern for the performance and scalability of these mechanisms. Even LBAC, when used with large complex queries, has been shown to impose minimal impact in terms of additional processing overhead. Nevertheless, concerns around manageability (Bertino & Sandhu, 2005) and completeness (Wang et al., 2007) should still be considered in the design of Web-based applications that connect to relational database management systems employing fine-grained access control.

Recommendations

The author's experimental work was performed using small-scale servers with single-core processors that were manufactured nearly a decade ago. While this server architecture is well documented in the literature, particularly with reference to evaluation of fine-grained access control mechanisms, it represents a simplistic approximation of the multi-core, multi-processor servers currently used to host large relational database management systems. There is a need for future research that models performance of large-scale relational database management systems in a multi-core, multi-processor environment. Hill and Marty (2008) describe some of the challenges facing database professionals and system architects tasked with deploying applications to multi-core, multi-processor servers. Determining scalability is paramount among those concerns.

A problem faced by the author was the significant effort required to configure, validate, and execute benchmarks to evaluate the performance overhead associated with fine-grained access control. Keeton and Patterson (2000) describe the benefits of using microbenchmarks for studying database workloads. Among the benefits of microbenchmarks are the requirements for less hardware, less configuration, and simpler database deployments. Properly configured microbenchmarks are particularly useful if they provide the same resource usage patterns as larger, better-documented workloads such as the TPC-W. Additional database performance research using microbenchmarks would be extremely useful, particularly if the methodology was well documented and the benchmark software developed through the research was packaged for reuse by other researchers.

Summary

The FACE model was developed to provide a research based instrument to compare the relative scalability and performance of four common fine-grained access control mechanisms implemented within relational database management systems. A detailed description of the methodology employed in this study and the experimental data gathered has been provided. Analysis of data using sign tables provided a suitable approach for summarizing data and generating statistics to allow validation of experimental results. Recommendations have been advanced concerning future research that would make studies employing benchmarking easier to undertake, and easier to replicate, while at the same time being more generalizable to modern server environments.

Appendix A

TPC-W Authorization Views

```
CREATE VIEW authview.address AS
SELECT * FROM public.address
WHERE addr_user_role = CURRENT_USER;

CREATE VIEW authview.author AS
SELECT * FROM public.author
WHERE a_user_role = CURRENT_USER;

CREATE VIEW authview.cc_xacts AS
SELECT * FROM public.cc_xacts
WHERE cx_user_role = CURRENT_USER;

CREATE VIEW authview.customer AS
SELECT * FROM public.customer
WHERE c_user_role = CURRENT_USER;

CREATE VIEW authview.item AS
SELECT * FROM public.item
WHERE i_user_role = CURRENT_USER;

CREATE VIEW authview.order_line AS
SELECT * FROM public.order_line
WHERE ol_user_role = CURRENT_USER;

CREATE VIEW authview.orders AS
SELECT * FROM public.orders
WHERE o_user_role = CURRENT_USER;

CREATE VIEW authview.shopping_cart AS
SELECT * FROM public.shopping_cart
WHERE sc_user_role = CURRENT_USER;

CREATE VIEW authview.shopping_cart_line AS
SELECT * FROM public.shopping_cart_line
WHERE scl_user_role = CURRENT_USER;
```

Appendix B

Sample TPC-W Hippocratic Database Views

```
-- Author View
CREATE VIEW hippo.author AS
SELECT a_id,
  (CASE WHEN a_p1=1
            THEN a_fname ELSE NULL END) AS a_fname,  a_lname,
  (CASE WHEN a_p2=1
            THEN a_mname ELSE NULL END) AS a_mname,
  (CASE WHEN a_p3=1
            THEN a_dob ELSE NULL END) AS a_dob,
     a_bio, a_p1, a_p2, a_p3, a_user_role, a_sec_label
FROM author;

-- Customer View
CREATE VIEW hippo.customer AS
SELECT c_id,
  (CASE WHEN c_p1=1
            THEN c_uname ELSE NULL END) AS c_uname,
  (CASE WHEN c_p2=1
            THEN c_passwd ELSE NULL END) AS c_passwd,
     c_fname, c_lname, c_addr_id, c_phone, c_email, c_since, c_last_login,
     c_login, c_expiration, c_discount, c_balance, c_ytd_pmt,
  (CASE WHEN c_p3=1
            THEN c_birthdate ELSE NULL END) AS c_birthdate,
     c_data, c_p1, c_p2, c_p3, c_user_role, c_sec_label
FROM customer;
```

Appendix C

TPC-W Label Based Access Control Views

```
CREATE VIEW lbac.address as
SELECT * FROM public.address
WHERE addr_sec_label IN (SELECT label FROM public.visiblelabels);

CREATE VIEW lbac.author as
SELECT * FROM public.author
WHERE a_sec_label IN (SELECT label FROM public.visiblelabels);

CREATE VIEW lbac.cc_xacts as
SELECT * FROM public.cc_xacts
WHERE cx_sec_label IN (SELECT label FROM public.visiblelabels);

CREATE VIEW lbac.customer as
SELECT * FROM public.customer
WHERE c_sec_label IN (SELECT label FROM public.visiblelabels);

CREATE VIEW lbac.item as
SELECT * FROM public.item
WHERE i_sec_label IN (SELECT label FROM public.visiblelabels);

CREATE VIEW lbac.order_line as
SELECT * FROM public.order_line
WHERE ol_sec_label IN (SELECT label FROM public.visiblelabels);

CREATE VIEW lbac.orders as
SELECT * FROM public.orders
WHERE o_sec_label IN (SELECT label FROM public.visiblelabels);

CREATE VIEW lbac.shopping_cart as
SELECT * FROM public.shopping_cart
WHERE sc_sec_label IN (SELECT label FROM public.visiblelabels);

CREATE VIEW lbac.shopping_cart_line as
SELECT * FROM public.shopping_cart_line
WHERE scl_sec_label IN (SELECT label FROM public.visiblelabels);
```

Appendix D

TPC-W Transparent Query Rewrite Views

```
CREATE OR REPLACE VIEW tqr.address AS
SELECT * FROM public.address
WHERE address.addr_tqr_ctx::text = app_context();

CREATE VIEW tqr.author AS
SELECT * FROM public.author
WHERE author.a_tqr_ctx::text = app_context();

CREATE VIEW tqr.cc_xacts AS
SELECT * FROM public.cc_xacts
WHERE cc_xacts.cx_tqr_ctx::text = app_context();

CREATE VIEW tqr.customer AS
SELECT * FROM public.customer
WHERE customer.c_tqr_ctx::text = app_context();

CREATE VIEW tqr.item AS
SELECT * FROM public.item
WHERE item.i_tqr_ctx::text = app_context();

CREATE VIEW tqr.order_line AS
SELECT * FROM public.order_line
WHERE order_line.ol_tqr_ctx::text = app_context();

CREATE VIEW tqr.orders AS
SELECT * FROM public.orders
WHERE orders.o_tqr_ctx::text = app_context();

CREATE VIEW tqr.shopping_cart AS
SELECT * FROM public.shopping_cart
WHERE shopping_cart.sc_tqr_ctx::text = app_context();

CREATE VIEW tqr.shopping_cart_line AS
SELECT * FROM public.shopping_cart_line
WHERE shopping_cart_line.scl_tqr_ctx::text = app_context();
```

Appendix E

Sample TPC-W Benchmark Queries

```
-- TPC-W benchmark simple query
SELECT c_fname,c_lname
FROM authview.customer WHERE c_id = ${100K_C_ID}
FOR SHARE OF customer  NOWAIT


-- TPC-W benchmark complex query
SELECT *
FROM authview.item, authview.author
WHERE item.i_a_id = author.a_id
    AND item.i_id = ${100K_I_ID}
FOR SHARE OF item, author  NOWAIT


-- TPC-W benchmark simple update
UPDATE authview.shopping_cart
       SET sc_date = current_date
WHERE sc_id = ${100K_SC_ID}


-- TPC-W benchmark complex update
UPDATE authview.shopping_cart_line
       SET scl_qty = ${100K_RND_QTY}
WHERE scl_i_id = ${100K_SC_ID}
    AND scl_sc_id IN
          (SELECT scl_sc_id
           FROM authview.shopping_cart_line
           WHERE scl_i_id = ${100K_SC_ID})
```

Appendix F

Sample Wildlife Benchmark Queries

```
-- Wildlife benchmark simple query
SELECT species_common_name AS "SPECIES",
            ecoprovince_description AS "ECOPROVINCE",
            season_code AS "SEASON",
            area AS "AREA"
FROM (authview.bei_capability_ecoprov_summry
LEFT JOIN authview.bei_ecoprovinces
          ON bei_capability_ecoprov_summry.ecoprovince_code =
              bei_ecoprovinces.ecoprovince_code)
LEFT JOIN authview.bei_species_codes
          ON bei_capability_ecoprov_summry.species_code =
              bei_species_codes.species_code
WHERE bei_capability_ecoprov_summry.ecoprovince_code =
          (SELECT ecoprovince_code FROM authview.bei_ecoprovinces
           ORDER BY RANDOM() LIMIT 1)
    AND bei_capability_ecoprov_summry.caps_p1 = 1
FOR SHARE OF bei_capability_ecoprov_summry NOWAIT


-- Wildlife benchmark complex query
SELECT area AS "AREA",
            perimeter AS "PERIMETER",
          qbei_tag AS "ID",
          zone_code || subzone_code || variant_code || phase_code AS "BEC"
FROM  authview.bei_polygon_attributes
WHERE poly_id = ${RND_POLY_ID}
    AND bei_polygon_attributes.poly_p1 = 1
FOR SHARE OF bei_polygon_attributes NOWAIT
```

Appendix G

Transparent Query Rewrite Stored Procedure

```
-- VPD-like procedural language policy function to set user security context
-- Note: The variable class 'context' is defined in postgresql.conf

DROP FUNCTION app_context();

CREATE OR REPLACE FUNCTION app_context()
  RETURNS text AS $BODY$
  DECLARE usrctx text;
BEGIN
  -- Set value for variable class 'context' defined in postgresql.conf
  IF CURRENT_SETTING('context.usr') = '' THEN
    IF CURRENT_USER = 'postgres' THEN
      SET context.usr = '%';
    ELSIF CURRENT_USER = 'tqr' THEN
      SET context.usr = 'MANAGER' ;
    ELSIF CURRENT_USER = 'benchmark' THEN
      SET context.usr = 'CLERK';
    ELSE
      SET context.usr = '-';
    END IF;
  END IF;
  -- User security context is memory resident after first iteration
  -- of stored procedure
  usrctx := CURRENT_SETTING('context.usr');
  RETURN usrctx;
END;
$BODY$ LANGUAGE plpgsql;

ALTER FUNCTION app_context()
  OWNER TO postgres;
```

Appendix H

TPC-W Sample Query Implemented Under Apache Jmeter

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="2.1">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="TPC-
W_AUTHVIEW_1M_SELECT_800" enabled="true">
      <elementProp name="TestPlan.user_defined_variables" elementType="Arguments"
guiclass="ArgumentsPanel" testclass="Arguments" testname="User Defined Variables"
enabled="true">
        <collectionProp name="Arguments.arguments">
          <elementProp name="EBS" elementType="Argument">
            <stringProp name="Argument.name">EBS</stringProp>
            <stringProp name="Argument.value">800</stringProp>
            <stringProp name="Argument.metadata">=</stringProp>
          </elementProp>
        </collectionProp>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
      <boolProp name="TestPlan.serialize_threadgroups">true</boolProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <stringProp name="TestPlan.comments"></stringProp>
    </TestPlan>
    <hashTree>
      <ResultCollector guiclass="SummaryReport" testclass="ResultCollector"
testname="Summary Report" enabled="true">
        <boolProp name="ResultCollector.error_logging">false</boolProp>
        <objProp>
          <name>saveConfig</name>
          <value class="SampleSaveConfiguration">
            <time>true</time>
            <latency>true</latency>
            <timestamp>true</timestamp>
            <success>true</success>
            <label>true</label>
            <code>true</code>
            <message>true</message>
            <threadName>true</threadName>
            <dataType>true</dataType>
```

```
            <encoding>false</encoding>
            <assertions>false</assertions>
            <subresults>false</subresults>
            <responseData>false</responseData>
            <samplerData>false</samplerData>
            <xml>false</xml>
            <fieldNames>true</fieldNames>
            <responseHeaders>false</responseHeaders>
            <requestHeaders>false</requestHeaders>
            <responseDataOnError>false</responseDataOnError>
<saveAssertionResultsFailureMessage>false</saveAssertionResultsFailureMessage>
            <assertionsResultsToSave>0</assertionsResultsToSave>
            <bytes>true</bytes>
            <threadCounts>true</threadCounts>
            <sampleCount>true</sampleCount>
          </value>
        </objProp>
        <stringProp name="filename"></stringProp>
        <boolProp name="ResultCollector.success_only_logging">true</boolProp>
      </ResultCollector>
      <hashTree/>
      <JDBCDataSource guiclass="TestBeanGUI" testclass="JDBCDataSource"
testname="JDBC Connection - 1M" enabled="true">
        <stringProp name="password">benchmark</stringProp>
        <stringProp name="timeout"></stringProp>
        <stringProp name="checkQuery">Select 1</stringProp>
        <stringProp name="trimInterval"></stringProp>
        <boolProp name="autocommit">true</boolProp>
        <stringProp name="poolMax"></stringProp>
        <stringProp name="driver">org.postgresql.Driver</stringProp>
        <stringProp name="connectionAge">5000</stringProp>
        <stringProp name="dataSource">1M</stringProp>
        <stringProp name="username">benchmark</stringProp>
        <boolProp name="keepAlive">false</boolProp>
        <stringProp
name="dbUrl">jdbc:postgresql://192.168.255.204/tpcw1M</stringProp>
        <stringProp name="TestPlan.comments">localhost</stringProp>
      </JDBCDataSource>
      <hashTree/>
      <RandomVariableConfig guiclass="TestBeanGUI"
testclass="RandomVariableConfig" testname="1M_C_ID" enabled="true">
        <stringProp name="maximumValue">864000</stringProp>
        <stringProp name="minimumValue">1</stringProp>
        <stringProp name="outputFormat"></stringProp>
        <boolProp name="perThread">true</boolProp>
        <stringProp name="randomSeed"></stringProp>
```

```
        <stringProp name="variableName">1M_C_ID</stringProp>
      </RandomVariableConfig>
      <hashTree/>
      <RandomVariableConfig guiclass="TestBeanGUI"
testclass="RandomVariableConfig" testname="1M_I_ID" enabled="true">
        <stringProp name="maximumValue">1000000</stringProp>
        <stringProp name="minimumValue">1</stringProp>
        <stringProp name="outputFormat"></stringProp>
        <boolProp name="perThread">true</boolProp>
        <stringProp name="randomSeed"></stringProp>
        <stringProp name="variableName">1M_I_ID</stringProp>
      </RandomVariableConfig>
      <hashTree/>
      <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup"
testname="Load Cache" enabled="true">
        <elementProp name="ThreadGroup.main_controller"
elementType="LoopController" guiclass="LoopControlPanel"
testclass="LoopController" testname="Loop Controller" enabled="true">
          <boolProp name="LoopController.continue_forever">false</boolProp>
          <stringProp name="LoopController.loops">15</stringProp>
        </elementProp>
        <stringProp name="ThreadGroup.num_threads">1000</stringProp>
        <stringProp name="ThreadGroup.ramp_time">0</stringProp>
        <longProp name="ThreadGroup.start_time">1309579657000</longProp>
        <longProp name="ThreadGroup.end_time">1309579657000</longProp>
        <boolProp name="ThreadGroup.scheduler">false</boolProp>
        <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
        <stringProp name="ThreadGroup.duration"></stringProp>
        <stringProp name="ThreadGroup.delay"></stringProp>
      </ThreadGroup>
      <hashTree>
        <JDBCSampler guiclass="TestBeanGUI" testclass="JDBCSampler"
testname="Dummy Request" enabled="true">
          <stringProp name="dataSource">1M</stringProp>
          <stringProp name="queryType">Select Statement</stringProp>
          <stringProp name="query">SELECT c_fname,c_lname
FROM public.customer WHERE c_id = ${1M_C_ID}
FOR SHARE OF customer  NOWAIT</stringProp>
          <stringProp name="queryArguments"></stringProp>
          <stringProp name="queryArgumentsTypes"></stringProp>
          <stringProp name="variableNames"></stringProp>
          <stringProp name="TestPlan.comments">Simple query</stringProp>
          <stringProp name="resultVariable"></stringProp>
        </JDBCSampler>
        <hashTree/>
```

```
      <JDBCSampler guiclass="TestBeanGUI" testclass="JDBCSampler"
testname="Dummy Request" enabled="true">
        <stringProp name="dataSource">1M</stringProp>
        <stringProp name="queryType">Select Statement</stringProp>
        <stringProp name="query">SELECT *
FROM authview.item, authview.author
WHERE item.i_a_id = author.a_id
AND item.i_id = ${1M_I_ID}
FOR SHARE OF item, author  NOWAIT
</stringProp>
        <stringProp name="queryArguments"></stringProp>
        <stringProp name="queryArgumentsTypes"></stringProp>
        <stringProp name="variableNames"></stringProp>
        <stringProp name="TestPlan.comments">Complex query</stringProp>
        <stringProp name="resultVariable"></stringProp>
      </JDBCSampler>
      <hashTree/>
    </hashTree>
    <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup"
testname="Restart PostgreSQL" enabled="true">
      <elementProp name="ThreadGroup.main_controller"
elementType="LoopController" guiclass="LoopControlPanel"
testclass="LoopController" testname="Loop Controller" enabled="true">
        <boolProp name="LoopController.continue_forever">false</boolProp>
        <stringProp name="LoopController.loops">1</stringProp>
      </elementProp>
      <stringProp name="ThreadGroup.num_threads">1</stringProp>
      <stringProp name="ThreadGroup.ramp_time">1</stringProp>
      <longProp name="ThreadGroup.start_time">1310052990000</longProp>
      <longProp name="ThreadGroup.end_time">1310052990000</longProp>
      <boolProp name="ThreadGroup.scheduler">false</boolProp>
      <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
      <stringProp name="ThreadGroup.duration"></stringProp>
      <stringProp name="ThreadGroup.delay"></stringProp>
    </ThreadGroup>
    <hashTree>
      <org.apache.jmeter.protocol.ssh.sampler.SSHSampler guiclass="TestBeanGUI"
testclass="org.apache.jmeter.protocol.ssh.sampler.SSHSampler" testname="SSH
Command" enabled="true">
        <stringProp name="command">/root/pgrestart.sh</stringProp>
        <stringProp name="hostname">192.168.255.204</stringProp>
        <stringProp name="password">8is2much</stringProp>
        <intProp name="port">22</intProp>
        <stringProp name="username">root</stringProp>
      </org.apache.jmeter.protocol.ssh.sampler.SSHSampler>
      <hashTree/>
```

```
      <JDBCSampler guiclass="TestBeanGUI" testclass="JDBCSampler"
testname="Dummy Request" enabled="true">
        <stringProp name="dataSource">1M</stringProp>
        <stringProp name="queryType">Select Statement</stringProp>
        <stringProp name="query">SELECT c_fname,c_lname
FROM public.customer WHERE c_id = ${1M_C_ID}
FOR SHARE OF customer  NOWAIT</stringProp>
        <stringProp name="queryArguments"></stringProp>
        <stringProp name="queryArgumentsTypes"></stringProp>
        <stringProp name="variableNames"></stringProp>
        <stringProp name="resultVariable"></stringProp>
      </JDBCSampler>
      <hashTree/>
      <JDBCSampler guiclass="TestBeanGUI" testclass="JDBCSampler"
testname="Dummy Request" enabled="true">
        <stringProp name="dataSource">1M</stringProp>
        <stringProp name="queryType">Select Statement</stringProp>
        <stringProp name="query">SELECT c_fname,c_lname
FROM public.customer WHERE c_id = ${1M_C_ID}
FOR SHARE OF customer  NOWAIT</stringProp>
        <stringProp name="queryArguments"></stringProp>
        <stringProp name="queryArgumentsTypes"></stringProp>
        <stringProp name="variableNames"></stringProp>
        <stringProp name="resultVariable"></stringProp>
      </JDBCSampler>
      <hashTree/>
      <JDBCSampler guiclass="TestBeanGUI" testclass="JDBCSampler"
testname="Dummy Request" enabled="true">
        <stringProp name="dataSource">1M</stringProp>
        <stringProp name="queryType">Select Statement</stringProp>
        <stringProp name="query">SELECT c_fname,c_lname
FROM public.customer WHERE c_id = ${1M_C_ID}
FOR SHARE OF customer  NOWAIT</stringProp>
        <stringProp name="queryArguments"></stringProp>
        <stringProp name="queryArgumentsTypes"></stringProp>
        <stringProp name="variableNames"></stringProp>
        <stringProp name="resultVariable"></stringProp>
      </JDBCSampler>
      <hashTree/>
    </hashTree>
    <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="1a"
enabled="true">
      <elementProp name="ThreadGroup.main_controller"
elementType="LoopController" guiclass="LoopControlPanel"
testclass="LoopController" testname="Loop Controller" enabled="true">
        <boolProp name="LoopController.continue_forever">false</boolProp>
```

```
        <stringProp name="LoopController.loops">1</stringProp>
      </elementProp>
      <stringProp name="ThreadGroup.num_threads">${EBS}</stringProp>
      <stringProp name="ThreadGroup.ramp_time">0</stringProp>
      <longProp name="ThreadGroup.start_time">1309579657000</longProp>
      <longProp name="ThreadGroup.end_time">1309579657000</longProp>
      <boolProp name="ThreadGroup.scheduler">false</boolProp>
      <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
      <stringProp name="ThreadGroup.duration"></stringProp>
      <stringProp name="ThreadGroup.delay"></stringProp>
    </ThreadGroup>
    <hashTree>
      <JDBCSampler guiclass="TestBeanGUI" testclass="JDBCSampler" testname="1M-
1a" enabled="true">
        <stringProp name="dataSource">1M</stringProp>
        <stringProp name="queryType">Select Statement</stringProp>
        <stringProp name="query">SELECT c_fname,c_lname
FROM authview.customer WHERE c_id = ${1M_C_ID}
FOR SHARE OF customer  NOWAIT</stringProp>
        <stringProp name="queryArguments"></stringProp>
        <stringProp name="queryArgumentsTypes"></stringProp>
        <stringProp name="variableNames"></stringProp>
        <stringProp name="TestPlan.comments">Simple query</stringProp>
        <stringProp name="resultVariable"></stringProp>
      </JDBCSampler>
      <hashTree/>
    </hashTree>
    <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="2a"
enabled="true">
      <elementProp name="ThreadGroup.main_controller"
elementType="LoopController" guiclass="LoopControlPanel"
testclass="LoopController" testname="Loop Controller" enabled="true">
        <boolProp name="LoopController.continue_forever">false</boolProp>
        <stringProp name="LoopController.loops">1</stringProp>
      </elementProp>
      <stringProp name="ThreadGroup.num_threads">${EBS}</stringProp>
      <stringProp name="ThreadGroup.ramp_time">0</stringProp>
      <longProp name="ThreadGroup.start_time">1309579657000</longProp>
      <longProp name="ThreadGroup.end_time">1309579657000</longProp>
      <boolProp name="ThreadGroup.scheduler">false</boolProp>
      <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
      <stringProp name="ThreadGroup.duration"></stringProp>
      <stringProp name="ThreadGroup.delay"></stringProp>
    </ThreadGroup>
    <hashTree>
```

```
      <JDBCSampler guiclass="TestBeanGUI" testclass="JDBCSampler" testname="1M-
2a" enabled="true">
        <stringProp name="dataSource">1M</stringProp>
        <stringProp name="queryType">Select Statement</stringProp>
        <stringProp name="query">SELECT *
FROM authview.item, authview.author
WHERE item.i_a_id = author.a_id
AND item.i_id = ${1M_I_ID}
FOR SHARE OF item, author  NOWAIT
</stringProp>
        <stringProp name="queryArguments"></stringProp>
        <stringProp name="queryArgumentsTypes"></stringProp>
        <stringProp name="variableNames"></stringProp>
        <stringProp name="TestPlan.comments">Complex query</stringProp>
        <stringProp name="resultVariable"></stringProp>
      </JDBCSampler>
      <hashTree/>
    </hashTree>
  </hashTree>
 </hashTree>
</jmeterTestPlan>
```

Appendix I

TPC-W Benchmark Graphs for 100,000 Items

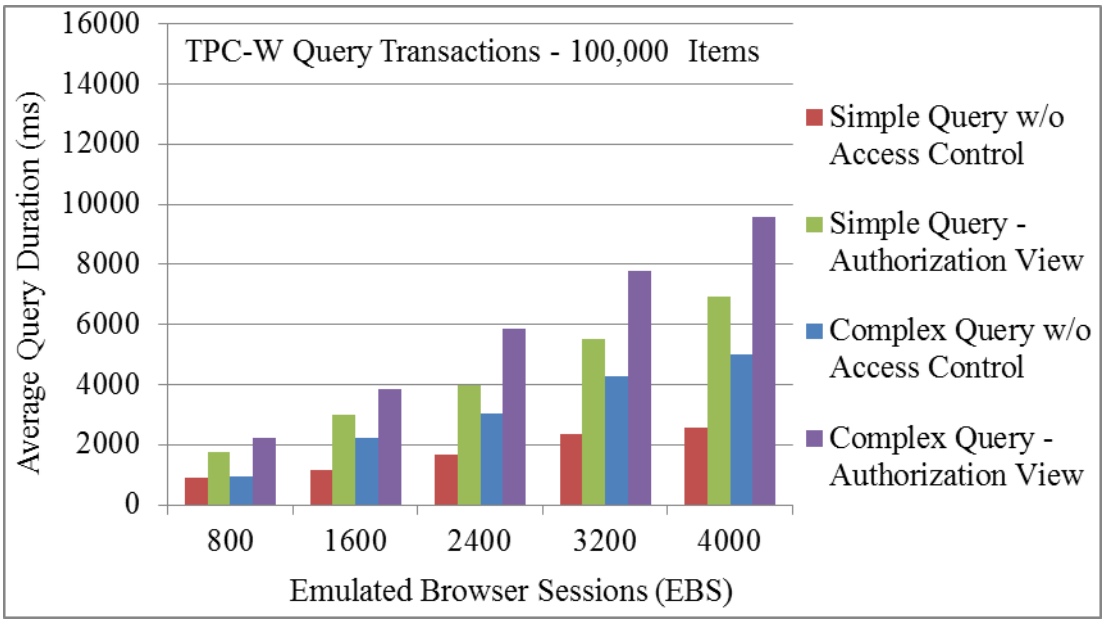

Figure 37. TPC-W Authorization View Query Transactions - 100,000 Items

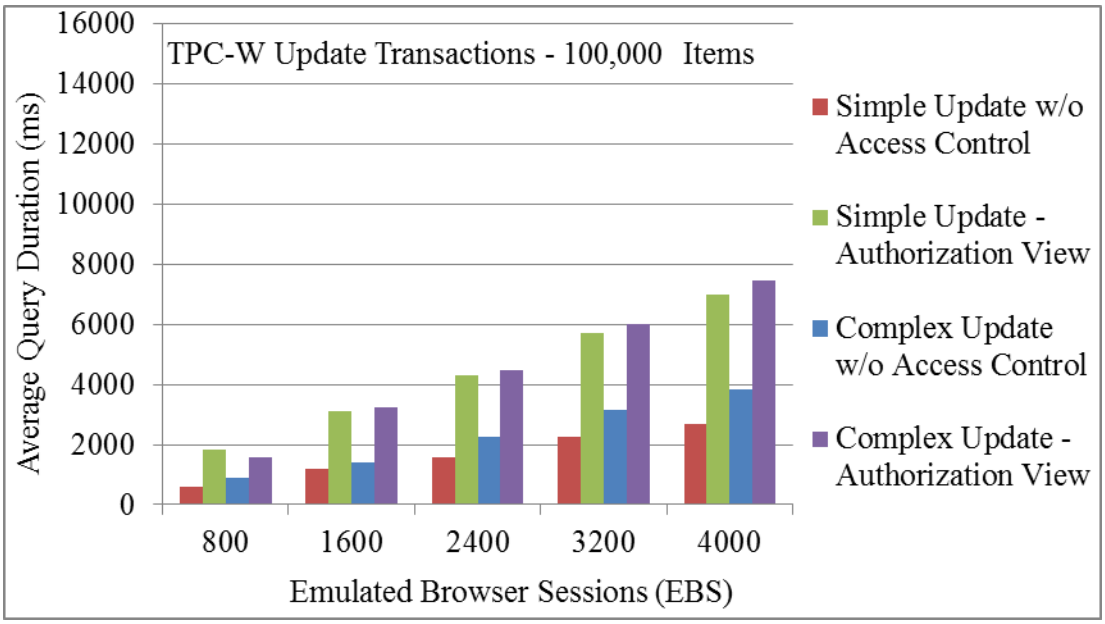

Figure 38. TPC-W Authorization View Update Transactions - 100,000 Items

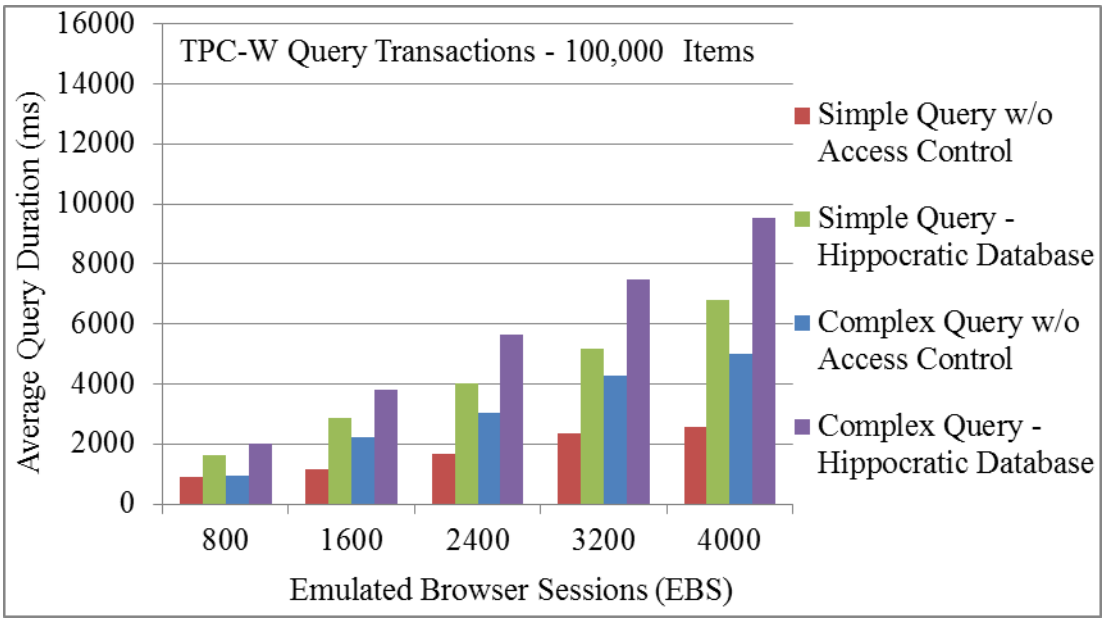

Figure 39. TPC-W Hippocratic Database Query Transactions - 100,000 Items

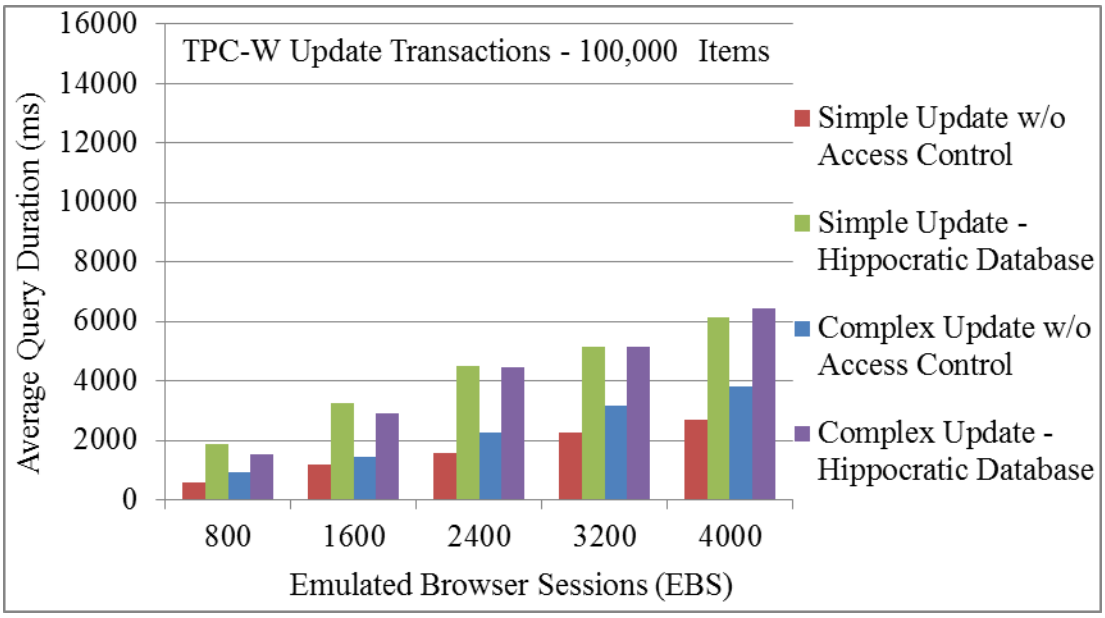

Figure 40. TPC-W Hippocratic Database Update Transactions - 100,000 Items

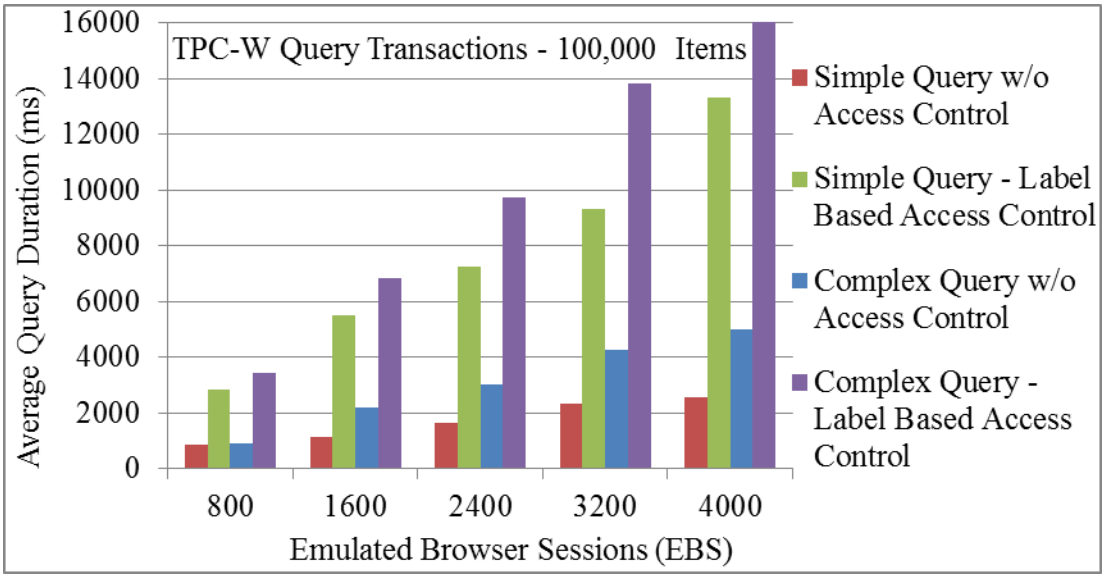

Figure 41. TPC-W Label Based Access Control Query Transactions - 100,000 Items

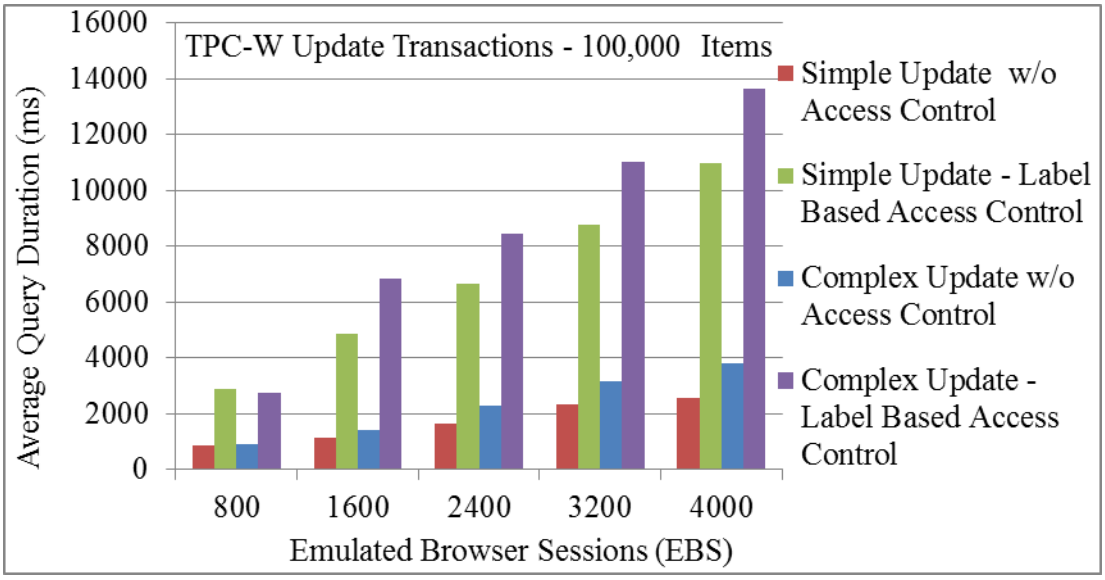

Figure 42. TPC-W Label Based Access Control Update Transactions - 100,000 Items

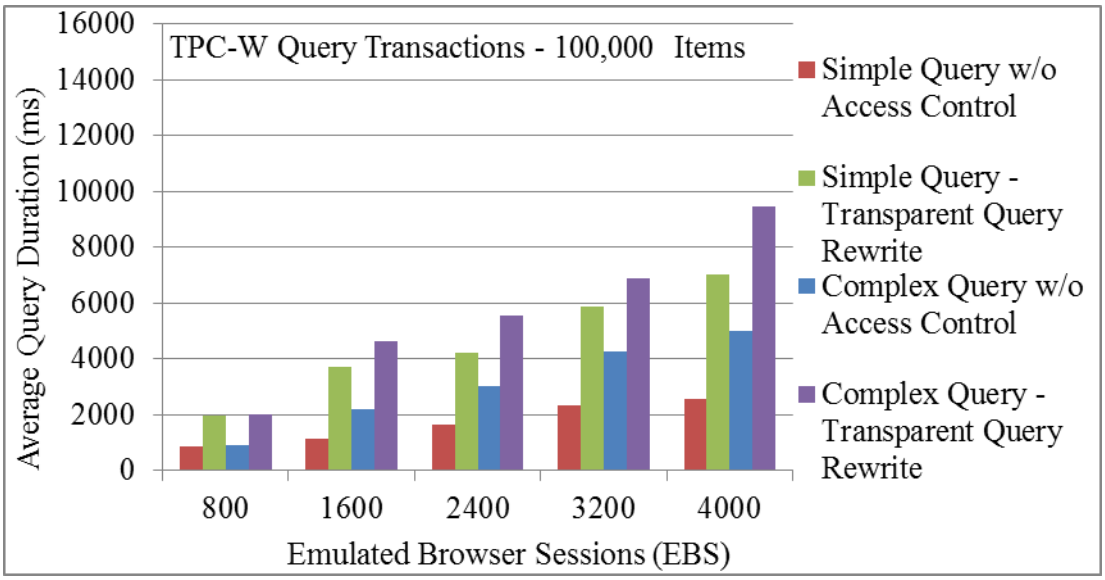

Figure 43. TPC-W Transparent Query Rewrite Query Transactions - 100,000 Items

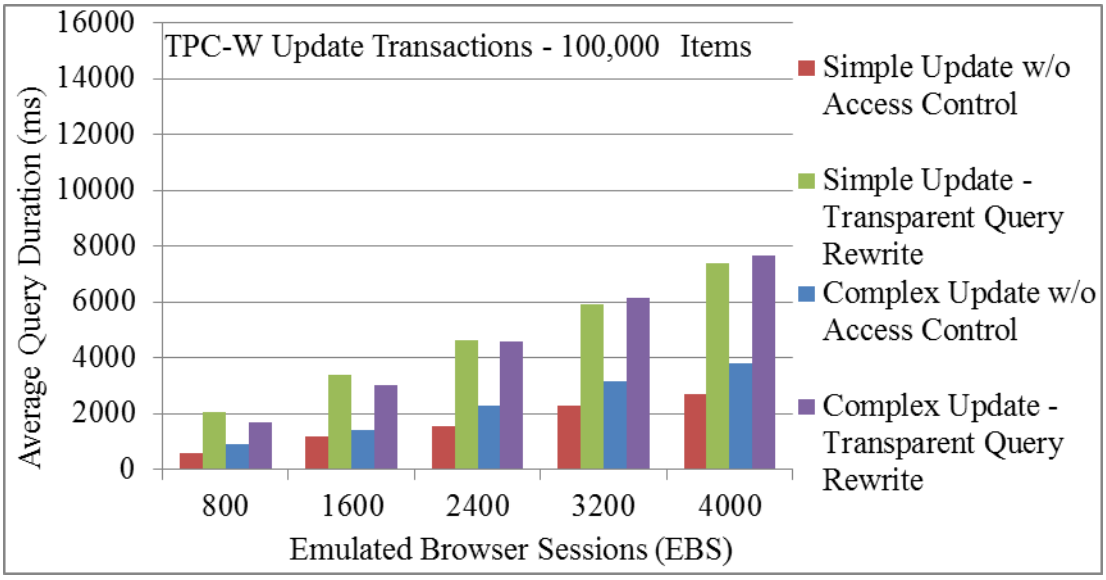

Figure 44. TPC-W Transparent Query Rewrite Update Transactions - 100,000 Items

Appendix J

TPC-W Benchmark Graphs for 1 Million Items

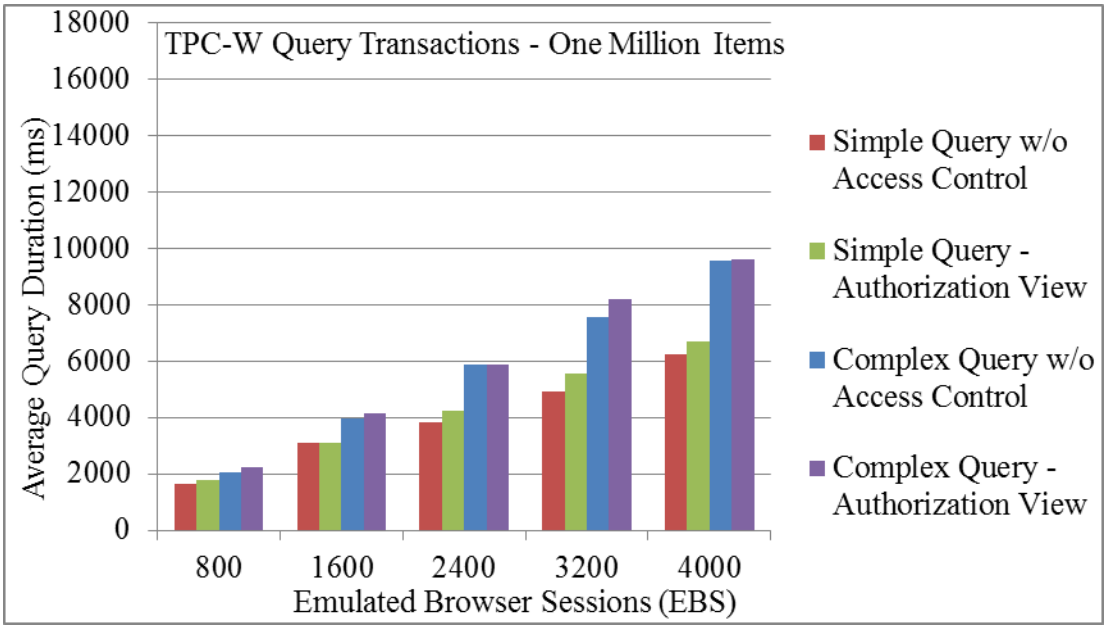

Figure 45. TPC-W Authorization View Query Transactions – One Million Items

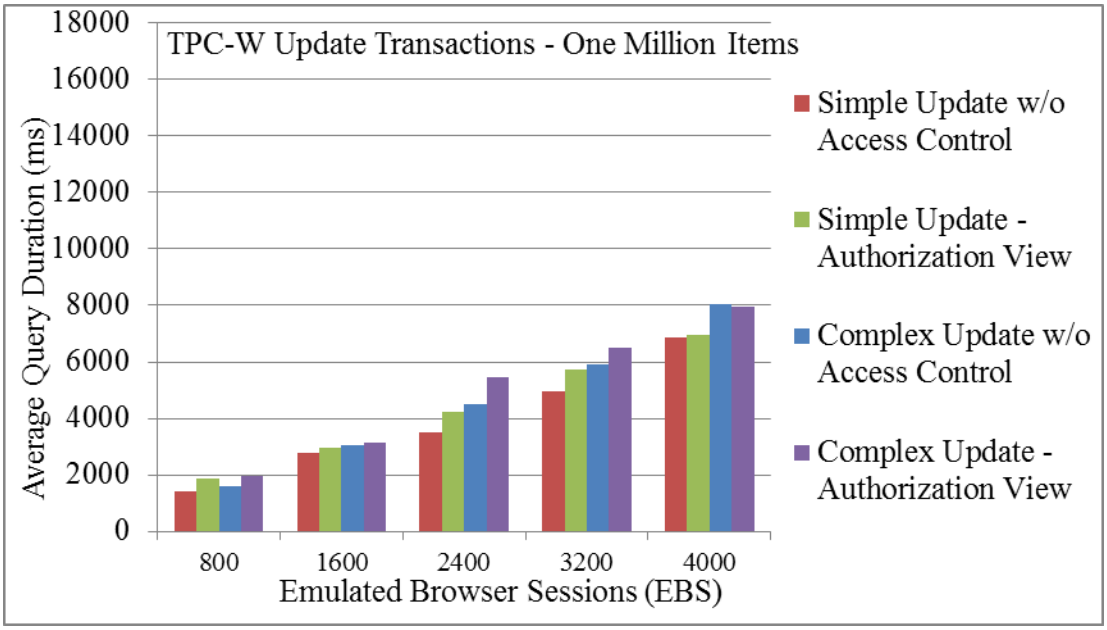

Figure 46. TPC-W Authorization View Update Transactions – One Million Items

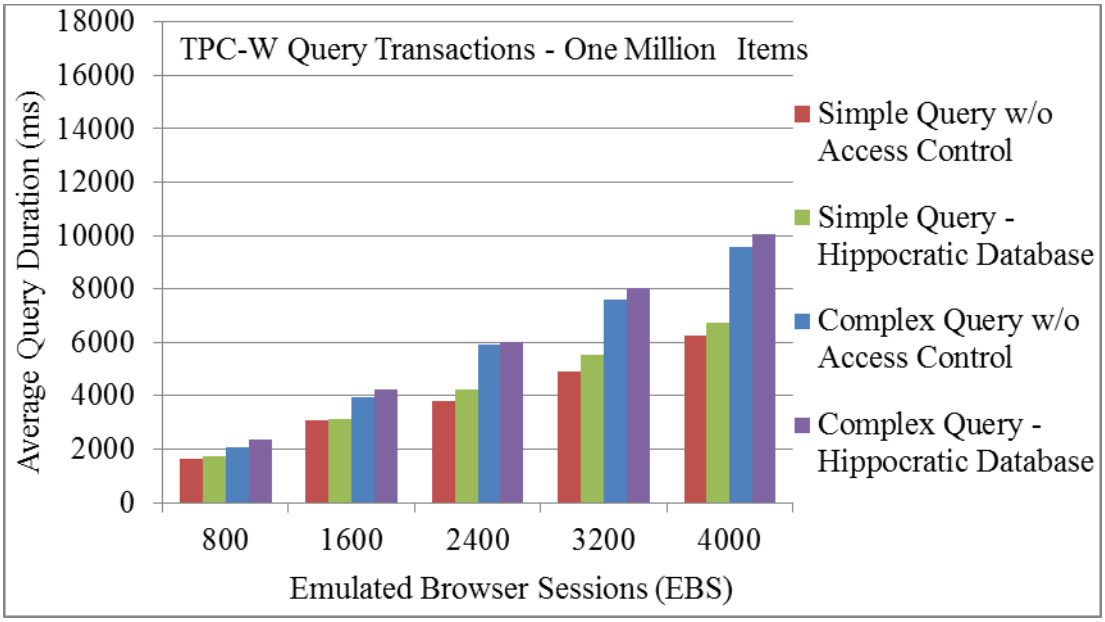

Figure 47. TPC-W Hippocratic Database Query Transactions – One Million Items

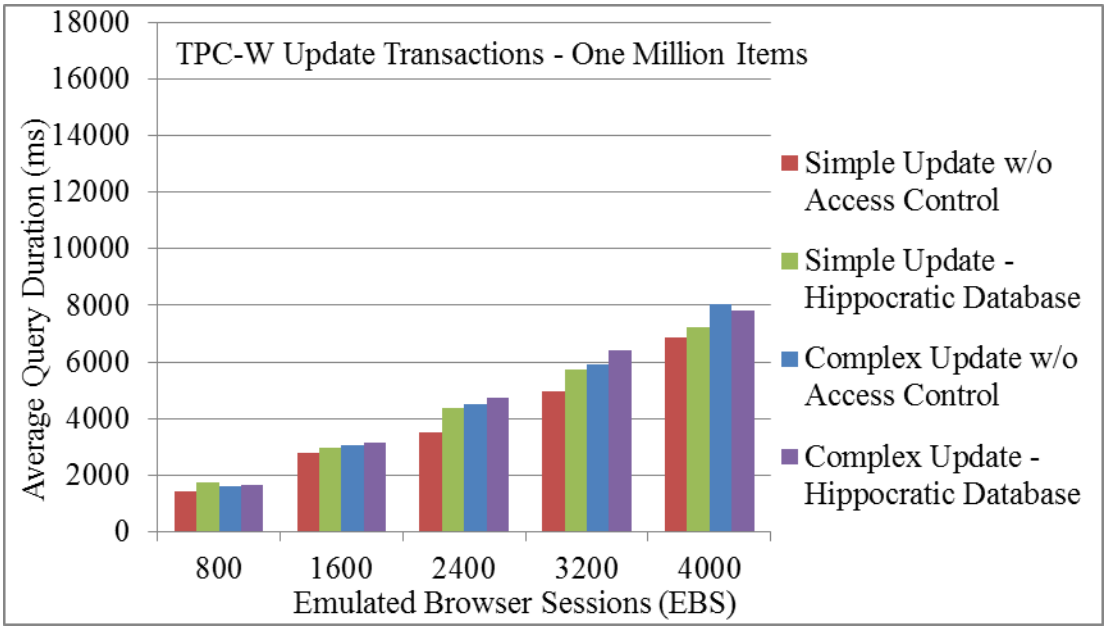

Figure 48. TPC-W Hippocratic Database Update Transactions – One Million Items

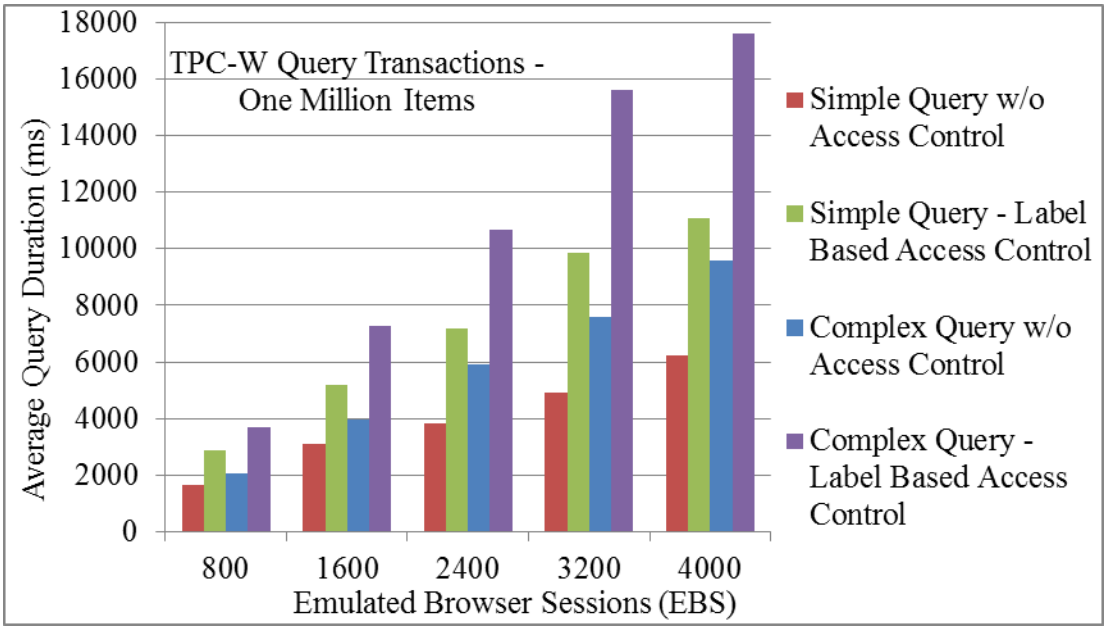

Figure 49. TPC-W Label Based Access Control Query Transactions – One Million Items

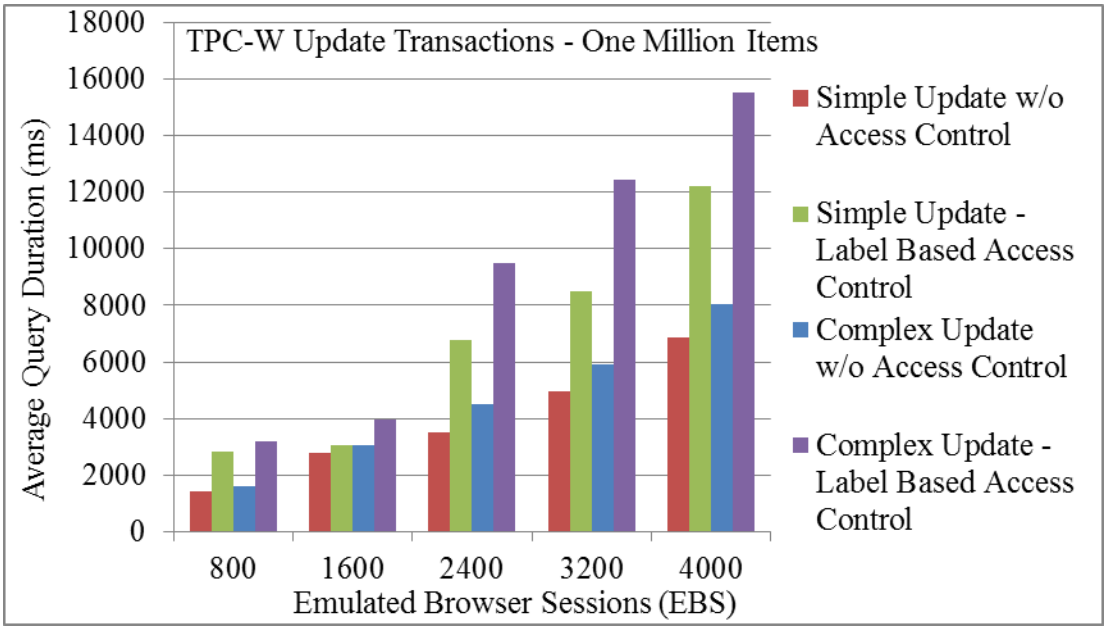

Figure 50. TPC-W Label Based Access Control Update Transactions – One Million Items

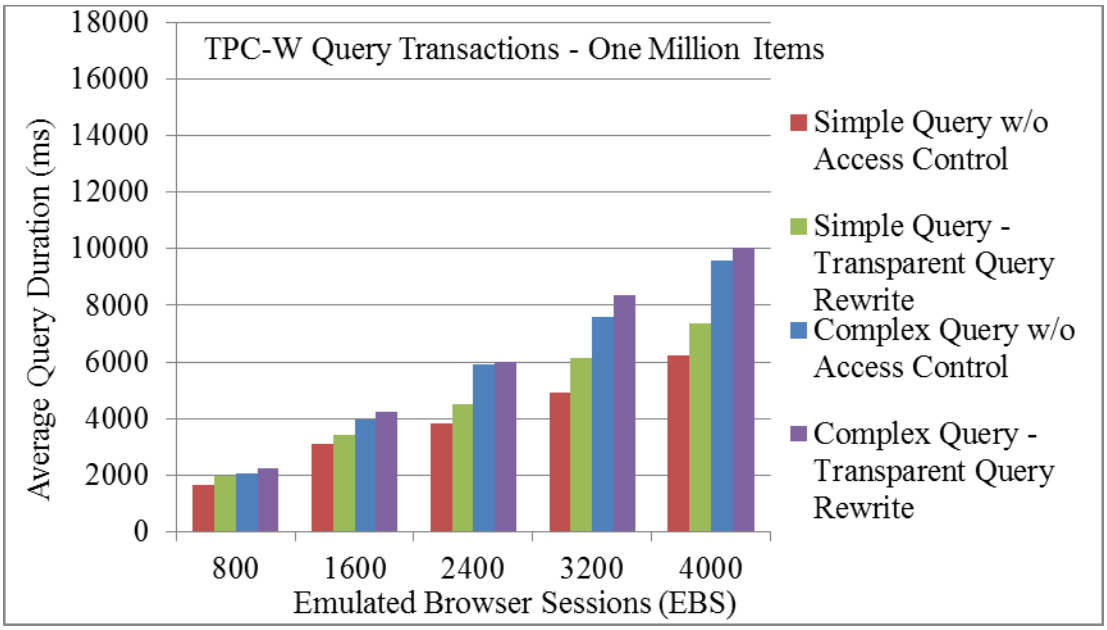

Figure 51. TPC-W Transparent Query Rewrite Query Transactions – One Million Items

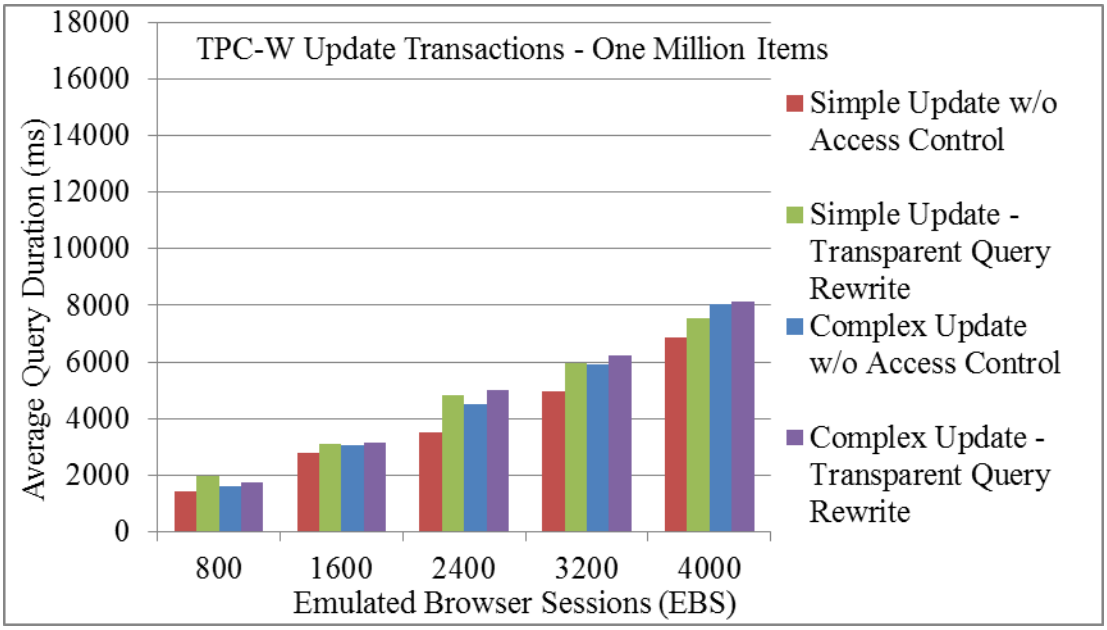

Figure 52. TPC-W Transparent Query Rewrite Update Transactions – One Million Items

Appendix K

TPC-W Benchmark Graphs for 10 Million Items

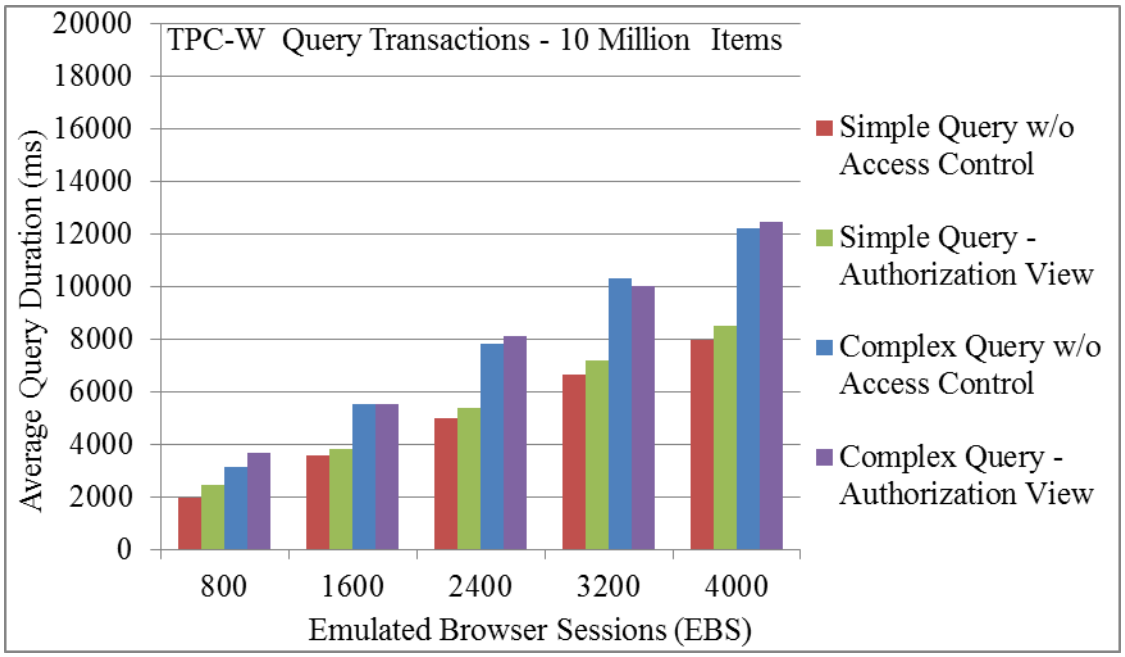

Figure 53. TPC-W Authorization View Query Transactions – 10 Million Items

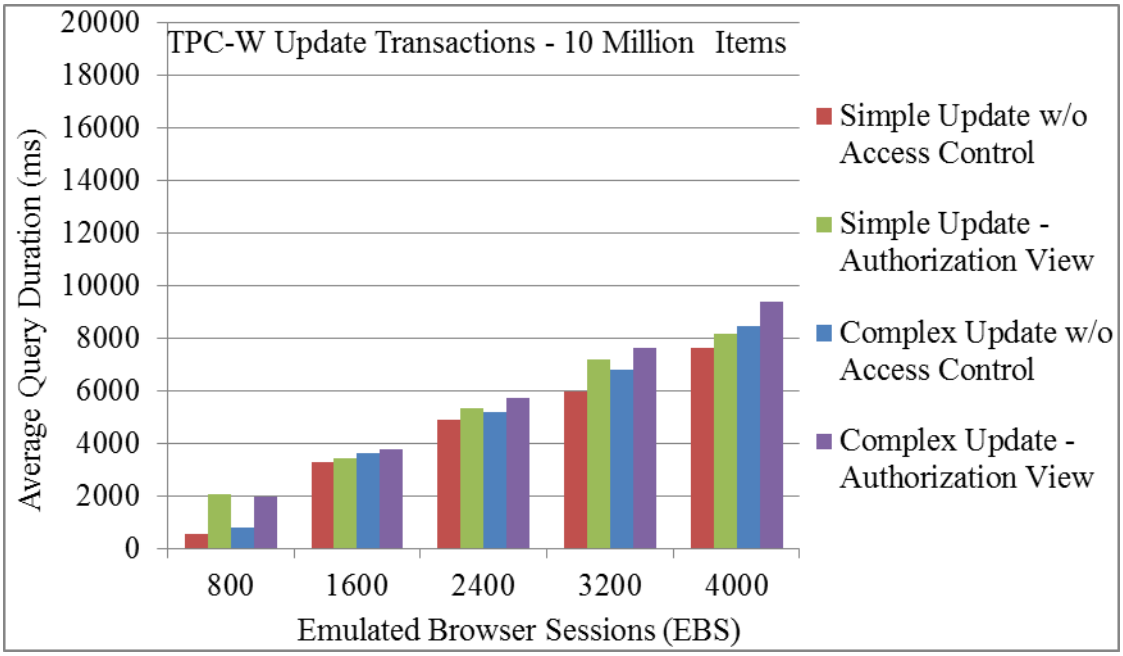

Figure 54. TPC-W Authorization View Update Transactions – 10 Million Items

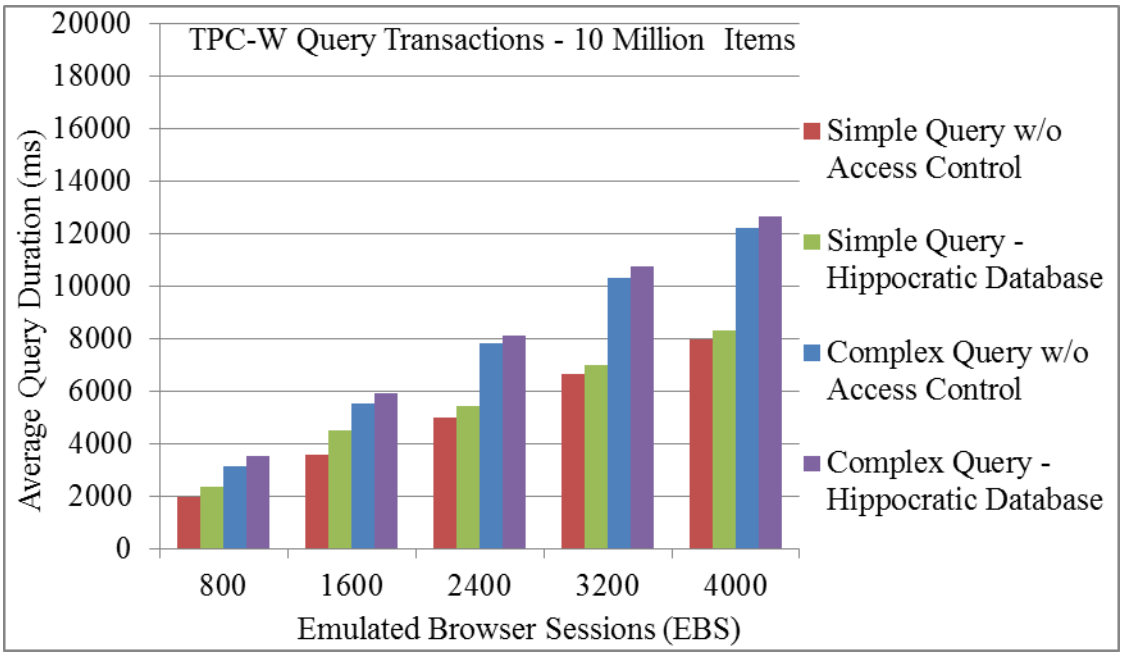

Figure 55. TPC-W Hippocratic Database Query Transactions – 10 Million Items

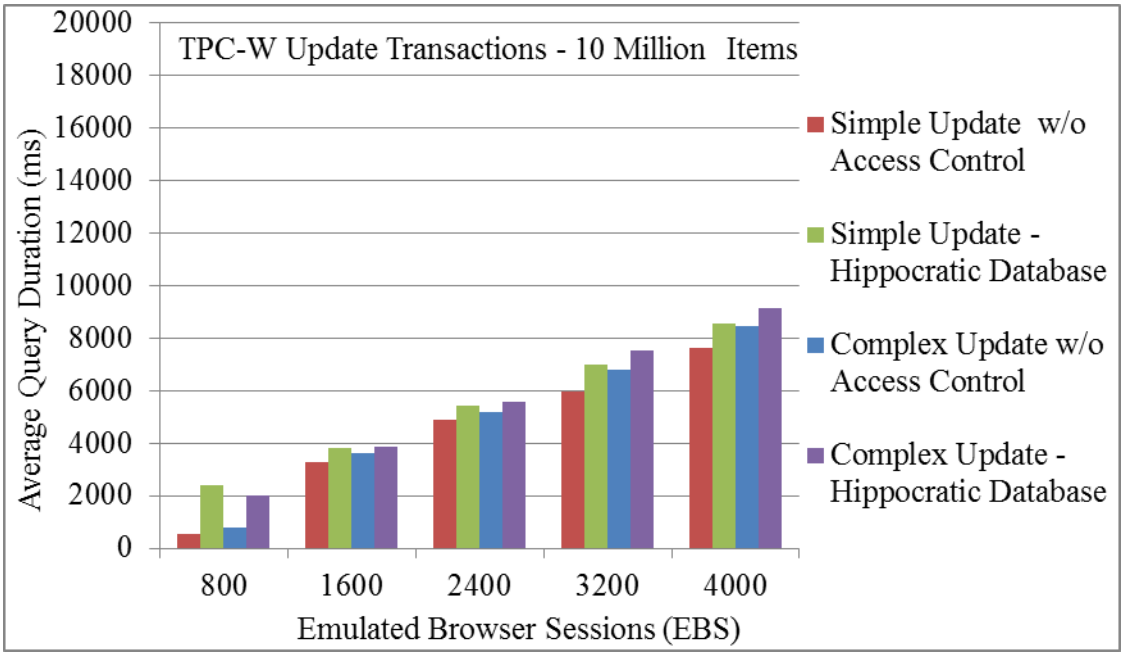

Figure 56. TPC-W Hippocratic Database Update Transactions – 10 Million Items

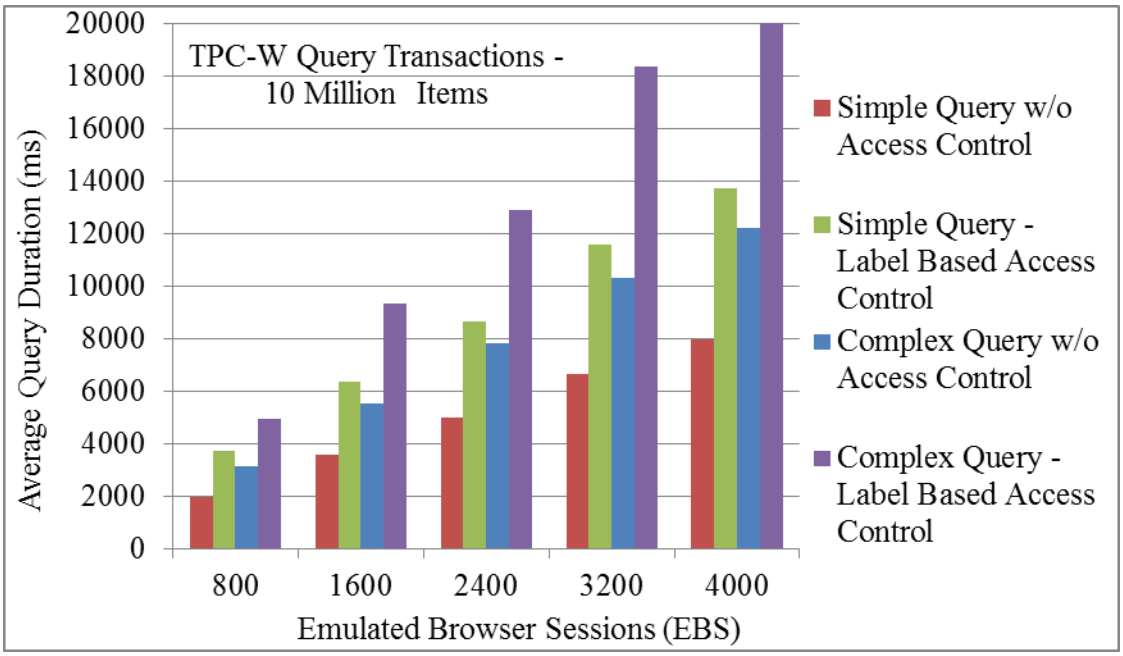

Figure 57. TPC-W Label Based Access Control Query Transactions – 10 Million Items

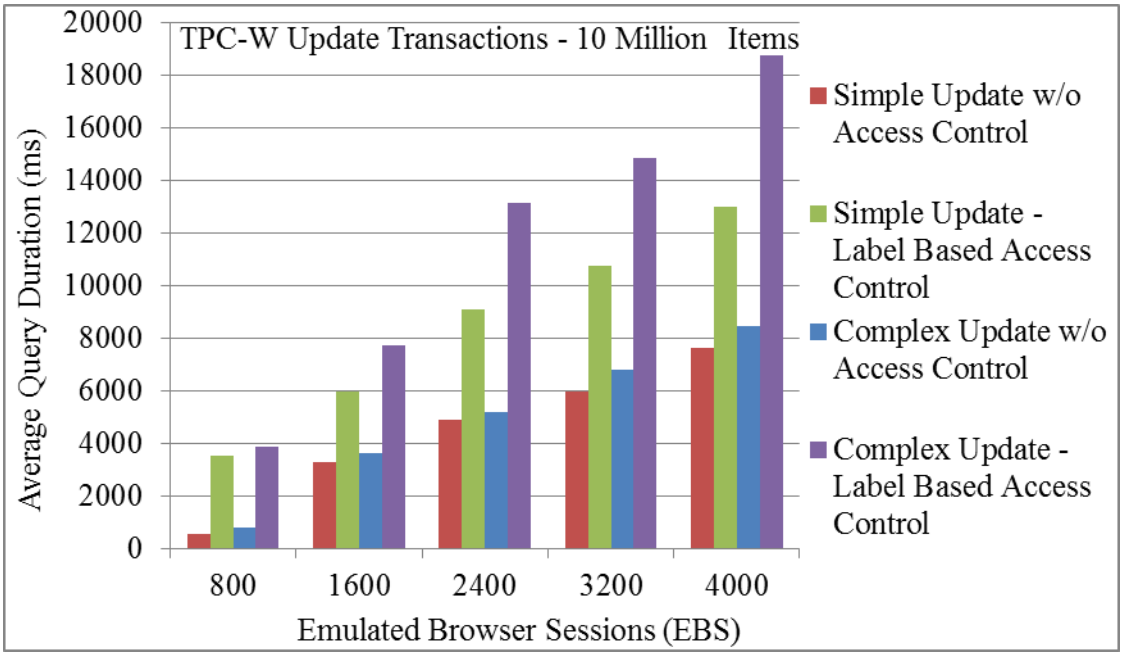

Figure 58. TPC-W Label Based Access Control Update Transactions – 10 Million Items

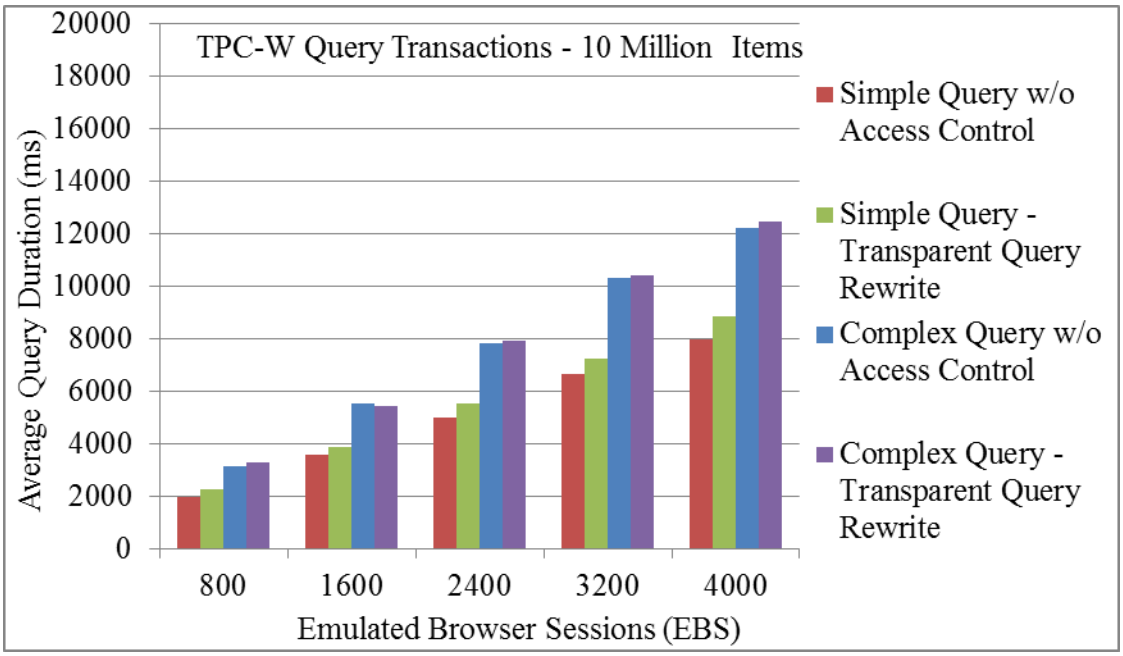

Figure 59. TPC-W Transparent Query Rewrite Query Transactions – 10 Million Items

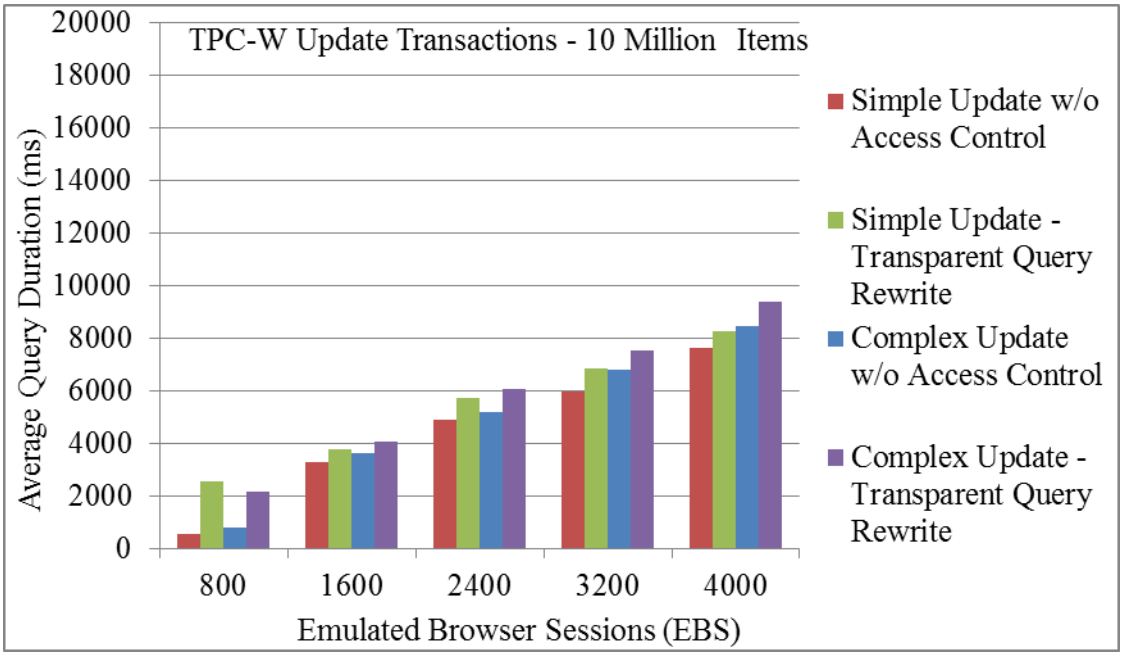

Figure 60. TPC-W Transparent Query Rewrite Update Transactions – 10 Million Items

Appendix L

Wildlife Data Benchmark Graphs

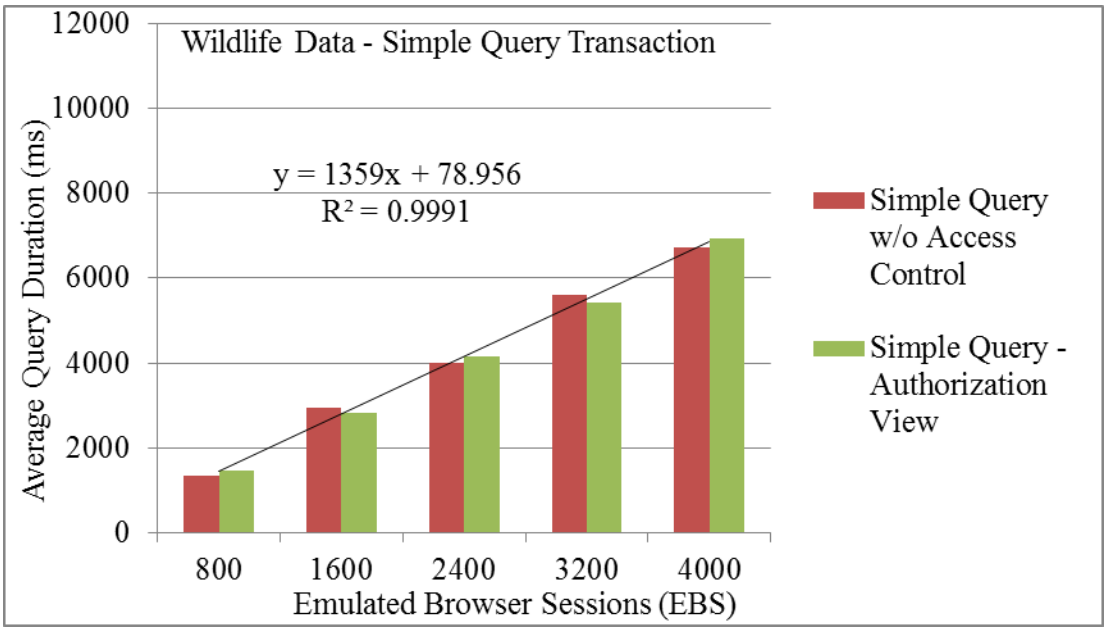

Figure 61. Wildlife Data Authorization Views - Simple Query Transactions

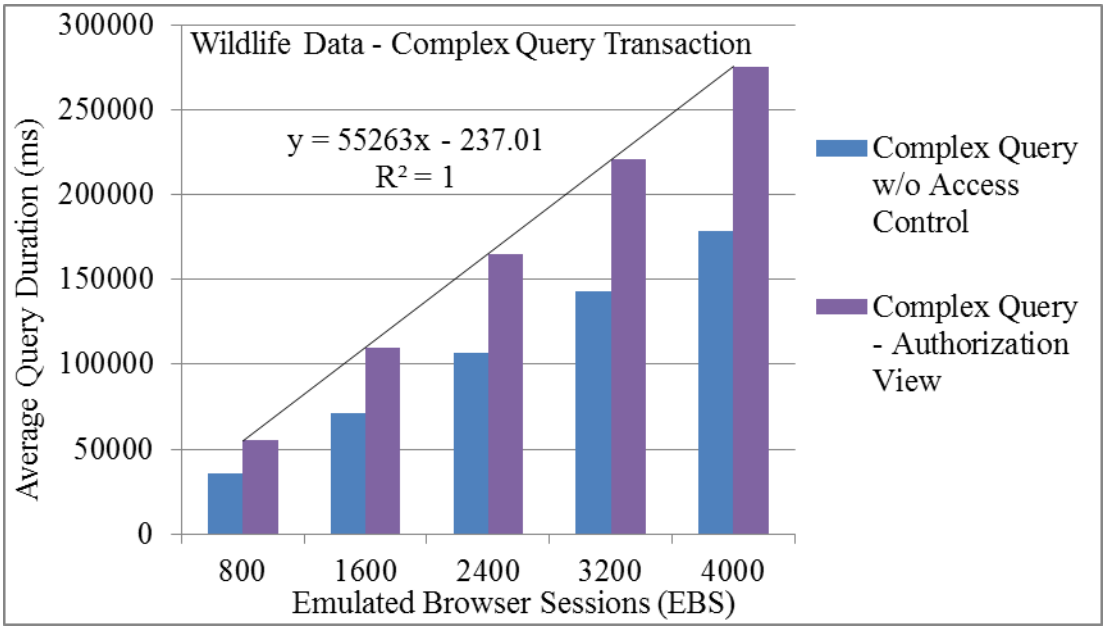

Figure 62. Wildlife Data Authorization Views - Complex Query Transactions

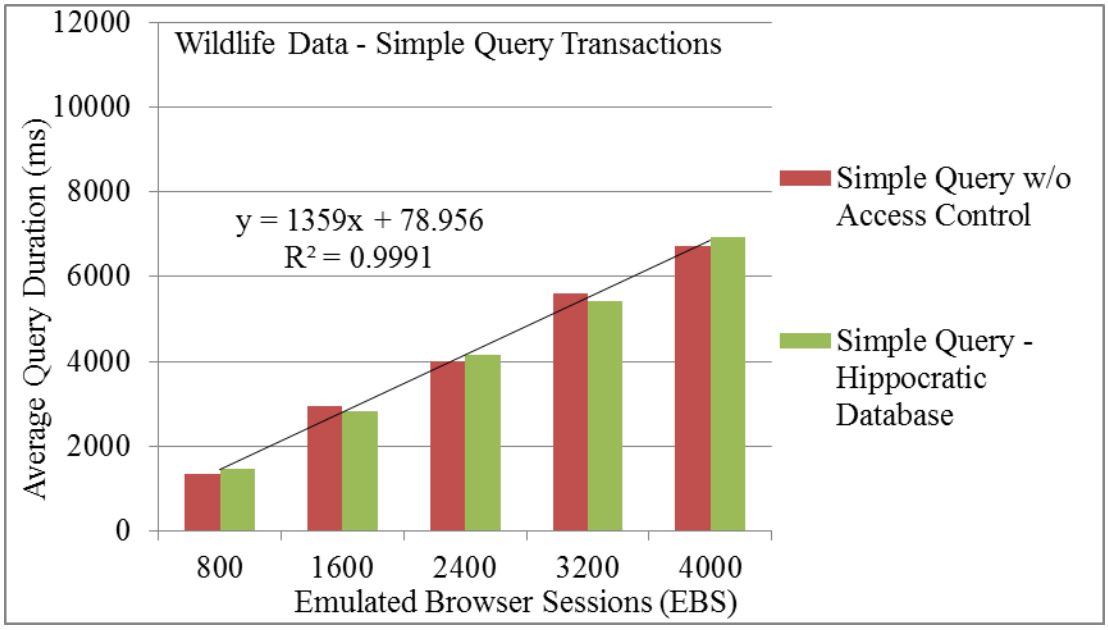

Figure 63. Wildlife Data Hippocratic Database - Simple Query Transactions

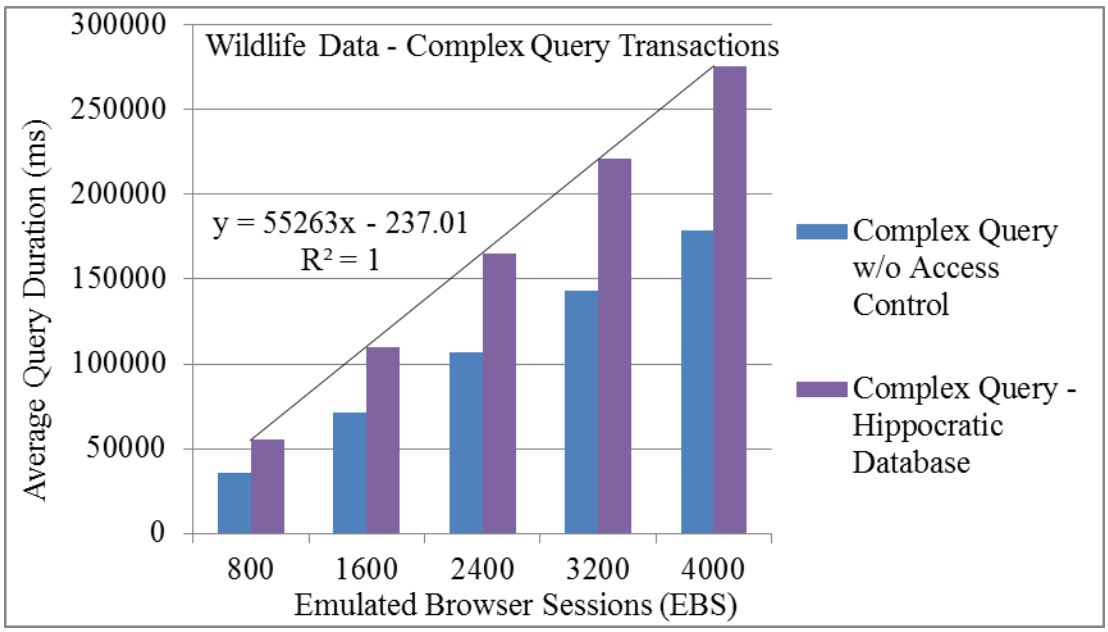

Figure 64. Wildlife Data Hippocratic Database - Complex Query Transactions

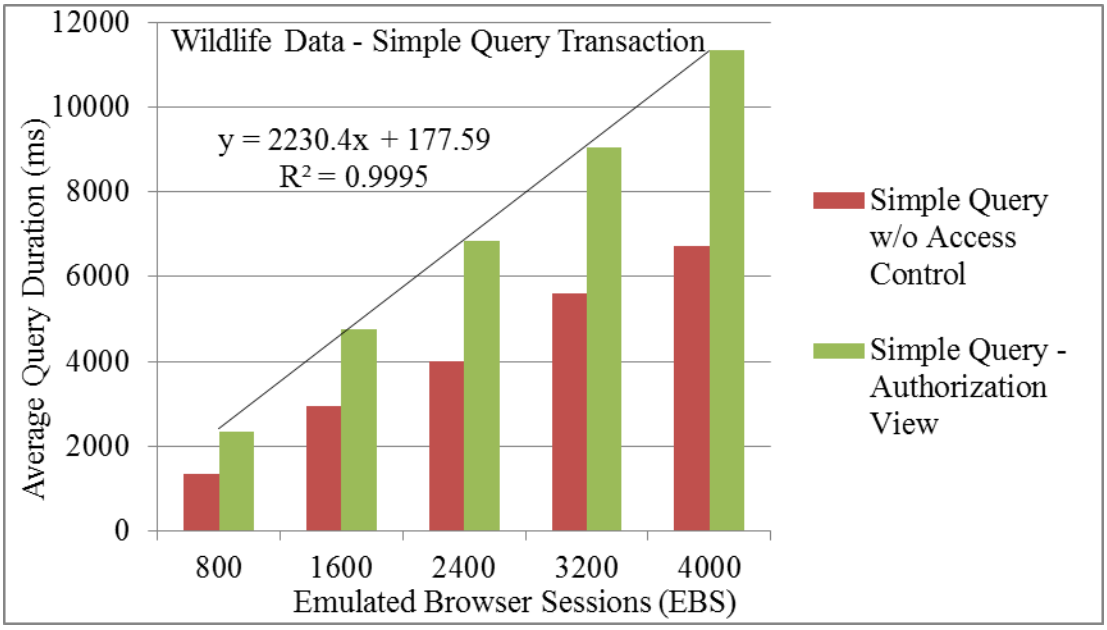

Figure 65. Wildlife Data Label Based Access Control - Simple Query Transactions

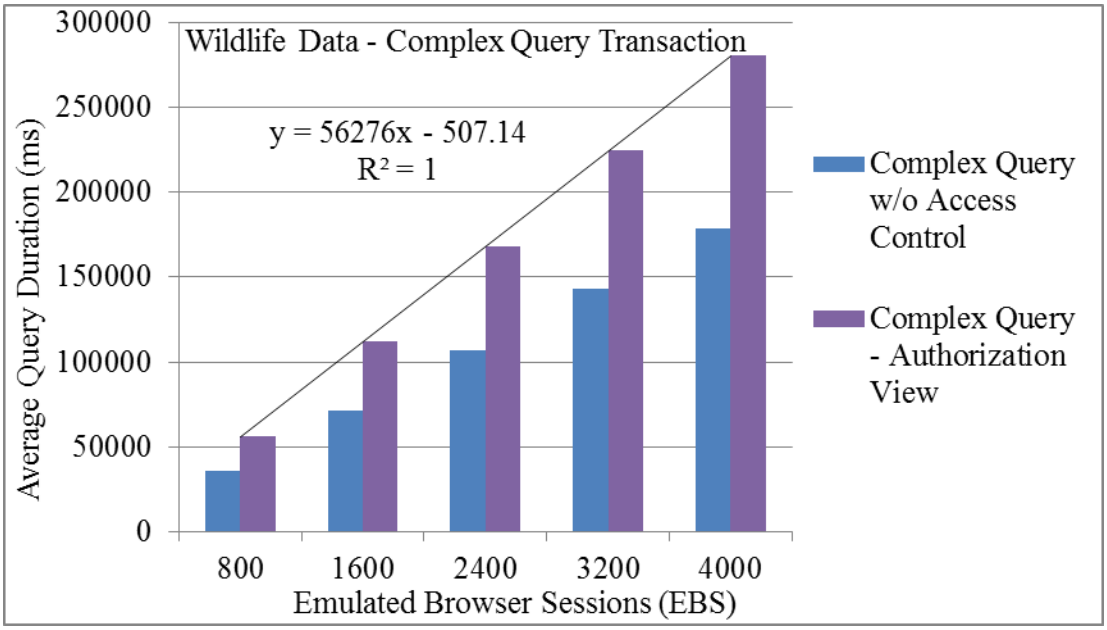

Figure 66. Wildlife Data Label Based Access Control - Complex Query Transactions

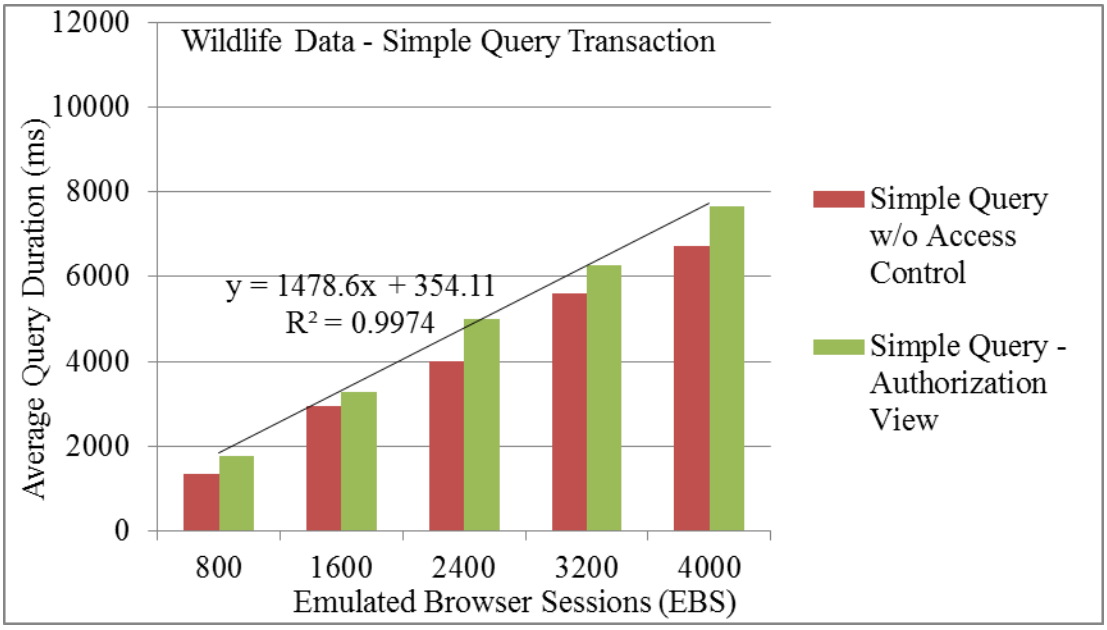

Figure 67. Wildlife Data Transparent Query Rewrite - Simple Query Transactions

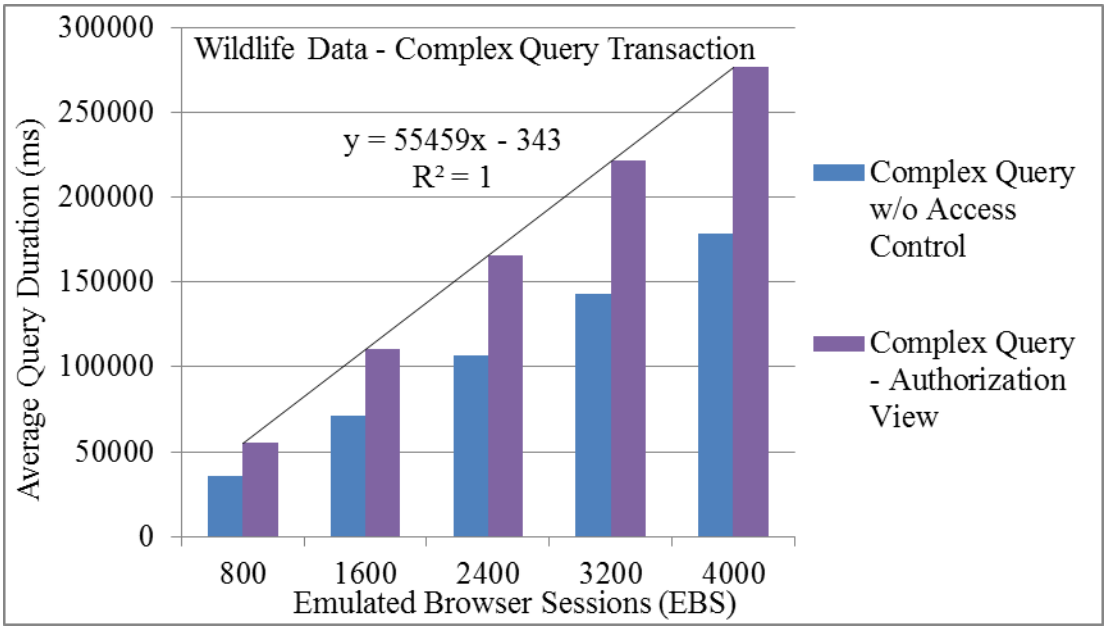

Figure 68. Wildlife Data Transparent Query Rewrite - Complex Query Transactions

Appendix M

TPC-W Benchmark Results for 100,000 Items

Table 12. TPC-W Authorization View – 100,000 Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	877	SST	7169266.92
1	1	-1	-1	-1	-1	1	1	1761	SSY	49964480.00
1	-1	1	-1	-1	1	-1	1	930	SS0	42795213.08
1	1	1	-1	1	-1	-1	-1	2207	S_e	34.64
1	-1	-1	1	1	-1	-1	1	582	S_{qi}	7.07
1	1	-1	1	-1	1	-1	-1	1850		
1	-1	1	1	-1	-1	1	-1	908		
1	1	1	1	1	1	1	1	1567		
10682.72	4087.597	541.956	-866.931	-216.924	-232.337	-455.299	-1002.38	*Totals*		

Table 13. TPC-W Authorization View – 100,000 Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1153	SST	22948905.21
1	1	-1	-1	-1	-1	1	1	2975	SSY	160284725.00
1	-1	1	-1	-1	1	-1	1	2216	SS0	137335819.79
1	1	1	-1	1	-1	-1	-1	3842	S_e	57.23
1	-1	-1	1	1	-1	-1	1	1185	S_{qi}	11.68
1	1	-1	1	-1	1	-1	-1	3094		
1	-1	1	1	-1	-1	1	-1	1423		
1	1	1	1	1	1	1	1	3248		
19137.11	7182.771	2323.486	-1235.89	-280.295	287.0689	-1538.1	111.4433	*Totals*		

Table 14. TPC-W Authorization View – 100,000 Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1651	SST	48708066.79
1	1	-1	-1	-1	-1	1	1	3995	SSY	324853556.00
1	-1	1	-1	-1	1	-1	1	3021	SS0	276145489.21
1	1	1	-1	1	-1	-1	-1	5852	S_e	71.01
1	-1	-1	1	1	-1	-1	1	1556	S_{qi}	14.50
1	1	-1	1	-1	1	-1	-1	4298		
1	-1	1	1	-1	-1	1	-1	2279		
1	1	1	1	1	1	1	1	4485		
27136.47	10123.27	4136.222	-1900.02	-49.79	-228.22	-2317.56	-1023.9	*Totals*		

Table 15. TPC-W Authorization View – 100,000 Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	2349	SST	80923582.59
1	1	-1	-1	-1	-1	1	1	5531	SSY	597571422.00
1	-1	1	-1	-1	1	-1	1	4285	SS0	516647839.41
1	1	1	-1	1	-1	-1	-1	7803	S_e	60.19
1	-1	-1	1	1	-1	-1	1	2274	S_{qi}	12.29
1	1	-1	1	-1	1	-1	-1	5713		
1	-1	1	1	-1	-1	1	-1	3163		
1	1	1	1	1	1	1	1	6000		
37117.75	12976.85	5384.143	-2817.8	-265	-423.161	-3030.46	-938.592	*Totals*		

Table 16. TPC-W Authorization View – 100,000 Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	2580	SST	132550926.86
1	1	-1	-1	-1	-1	1	1	6932	SSY	894058242.00
1	-1	1	-1	-1	1	-1	1	4986	SS0	761507315.14
1	1	1	-1	1	-1	-1	-1	9584	S_e	75.97
1	-1	-1	1	1	-1	-1	1	2694	S_{qi}	15.51
1	1	-1	1	-1	1	-1	-1	7015		
1	-1	1	1	-1	-1	1	-1	3824		
1	1	1	1	1	1	1	1	7448		
45063.14	16894.6	6622.579	-3100.83	-451.325	-1006.34	-3495.19	-942.825	*Totals*		

Table 17. TPC-W Hippocratic Database – 100,000 Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	877	SST	6070678.91
1	1	-1	-1	-1	-1	1	1	1613	SSY	46316346.33
1	-1	1	-1	-1	1	-1	1	930	SS0	40245667.42
1	1	1	-1	1	-1	-1	-1	2028	S_e	34.38
1	-1	-1	1	1	-1	-1	1	582	S_{qi}	7.02
1	1	-1	1	-1	1	-1	-1	1894		
1	-1	1	1	-1	-1	1	-1	908		
1	1	1	1	1	1	1	1	1526		
10359.62	3764.497	427.172	-538.739	-331.708	95.85499	-509.088	-1056.17	*Totals*		

Table 18. TPC-W Hippocratic Database – 100,000 Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1153	SST	21620653.37
1	1	-1	-1	-1	-1	1	1	2860	SSY	154416176.00
1	-1	1	-1	-1	1	-1	1	2216	SS0	132795522.63
1	1	1	-1	1	-1	-1	-1	3812	S_e	42.29
1	-1	-1	1	1	-1	-1	1	1185	S_{qi}	8.63
1	1	-1	1	-1	1	-1	-1	3249		
1	-1	1	1	-1	-1	1	-1	1423		
1	1	1	1	1	1	1	1	2921		
18818.11	6863.777	1925.36	-1263.65	-678.421	259.3119	-2104.52	-454.976	*Totals*		

Table 19. TPC-W Hippocratic Database – 100,000 Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1651	SST	46902982.62
1	1	-1	-1	-1	-1	1	1	4031	SSY	323133974.00
1	-1	1	-1	-1	1	-1	1	3021	SS0	276230991.38
1	1	1	-1	1	-1	-1	-1	5642	S_e	50.37
1	-1	-1	1	1	-1	-1	1	1556	S_{qi}	10.28
1	1	-1	1	-1	1	-1	-1	4510		
1	-1	1	1	-1	-1	1	-1	2279		
1	1	1	1	1	1	1	1	4451		
27140.67	10127.47	3645.373	-1548.19	-540.64	123.6049	-2315.77	-1022.11	*Totals*		

Table 20. TPC-W Hippocratic Database – 100,000 Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	2349	SST	64491625.64
1	1	-1	-1	-1	-1	1	1	5157	SSY	524342516.30
1	-1	1	-1	-1	1	-1	1	4285	SS0	459850890.66
1	1	1	-1	1	-1	-1	-1	7485	S_e	33.73
1	-1	-1	1	1	-1	-1	1	2274	S_{qi}	6.88
1	1	-1	1	-1	1	-1	-1	5162		
1	-1	1	1	-1	-1	1	-1	3163		
1	1	1	1	1	1	1	1	5144		
35018.12	10877.23	5134.472	-3533.91	-514.671	-1139.27	-3392.52	-1300.66	*Totals*		

Table 21, TPC-W Hippocratic Database – 100,000 Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	2580	SST	116141360.54
1	1	-1	-1	-1	-1	1	1	6785	SSY	809810975.00
1	-1	1	-1	-1	1	-1	1	4986	SS0	693669614.46
1	1	1	-1	1	-1	-1	-1	9548	S_e	62.36
1	-1	-1	1	1	-1	-1	1	2694	S_{qi}	12.73
1	1	-1	1	-1	1	-1	-1	6154		
1	-1	1	1	-1	-1	1	-1	3824		
1	1	1	1	1	1	1	1	6438		
43009.13	14840.6	6583.92	-4786.99	-489.984	-2692.5	-3755.18	-1202.82	*Totals*		

Table 22. TPC-W Label Based Access Control – 100,000 Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	877	SST	28639504.09
1	1	-1	-1	-1	-1	1	1	2824	SSY	114908075.00
1	-1	1	-1	-1	1	-1	1	930	SS0	86268570.91
1	1	1	-1	1	-1	-1	-1	3419	S_e	29.53
1	-1	-1	1	1	-1	-1	1	582	S_{qi}	6.03
1	1	-1	1	-1	1	-1	-1	2864		
1	-1	1	1	-1	-1	1	-1	908		
1	1	1	1	1	1	1	1	2763		
15167.38	8572.258	873.5342	-933.223	114.6539	-298.63	-421.178	-968.261	*Totals*		

Table 23. TPC-W Label Based Access Control – 100,000 Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1153	SST	133626515.24
1	1	-1	-1	-1	-1	1	1	5521	SSY	471981957.00
1	-1	1	-1	-1	1	-1	1	2216	SS0	338355441.76
1	1	1	-1	1	-1	-1	-1	6828	S_e	40.05
1	-1	-1	1	1	-1	-1	1	1185	S_{qi}	8.17
1	1	-1	1	-1	1	-1	-1	4865		
1	-1	1	1	-1	-1	1	-1	1423		
1	1	1	1	1	1	1	1	6846		
30038	18083.66	4589.587	-1399.11	1985.806	123.8519	-150.559	1498.987	*Totals*		

Table 24. TPC-W Label Based Access Control – 100,000 Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1651	SST	229313142.69
1	1	-1	-1	-1	-1	1	1	7265	SSY	847369950.00
1	-1	1	-1	-1	1	-1	1	3021	SS0	618056807.31
1	1	1	-1	1	-1	-1	-1	9725	S_e	50.59
1	-1	-1	1	1	-1	-1	1	1556	S_{qi}	10.33
1	1	-1	1	-1	1	-1	-1	6654		
1	-1	1	1	-1	-1	1	-1	2279		
1	1	1	1	1	1	1	1	8447		
40597.43	23584.23	6344.707	-2724.95	2158.695	-1053.16	-1313.88	-20.2266	*Totals*		

Table 25. TPC-W Label Based Access Control – 100,000 Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	2349	SST	412561435.04
1	1	-1	-1	-1	-1	1	1	9347	SSY	1549254372.00
1	-1	1	-1	-1	1	-1	1	4285	SS0	1136692936.96
1	1	1	-1	1	-1	-1	-1	13833	S_e	50.10
1	-1	-1	1	1	-1	-1	1	2274	S_{qi}	10.23
1	1	-1	1	-1	1	-1	-1	8784		
1	-1	1	1	-1	-1	1	-1	3163		
1	1	1	1	1	1	1	1	11022		
55056.16	30915.26	9548.824	-4570.88	3899.681	-2176.24	-3293.01	-1201.15	*Totals*		

Table 26. TPC-W Label Based Access Control – 100,000 Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	2580	SST	666198516.09
1	1	-1	-1	-1	-1	1	1	13320	SSY	2424978684.00
1	-1	1	-1	-1	1	-1	1	4986	SS0	1758780167.91
1	1	1	-1	1	-1	-1	-1	16440	S_e	65.25
1	-1	-1	1	1	-1	-1	1	2694	S_{qi}	13.32
1	1	-1	1	-1	1	-1	-1	10970		
1	-1	1	1	-1	-1	1	-1	3824		
1	1	1	1	1	1	1	1	13669		
68484.16	40315.63	9356.271	-6167.95	2282.367	-4073.46	-1697.55	854.8184	*Totals*		

Table 27. TPC-W Transparent Query Rewrite – 100,000 Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	877	SST	132233180.70
1	1	-1	-1	-1	-1	1	1	1960	SSY	177881808.00
1	-1	1	-1	-1	1	-1	1	930	SS0	45648627.30
1	1	1	-1	1	-1	-1	-1	2013	S_e	33.50
1	-1	-1	1	1	-1	-1	1	582	S_{qi}	6.84
1	1	-1	1	-1	1	-1	-1	2057		
1	-1	1	1	-1	-1	1	-1	908		
1	1	1	1	1	1	1	1	1706		
11033.12	4437.991	80.89547	-527.971	-677.985	106.6224	-129.256	-676.338	*Totals*		

Table 28. TPC-W Transparent Query Rewrite – 100,000 Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1153	SST	35329818.94
1	1	-1	-1	-1	-1	1	1	3732	SSY	196543248.00
1	-1	1	-1	-1	1	-1	1	2216	SS0	161213429.06
1	1	1	-1	1	-1	-1	-1	4613	S_e	44.20
1	-1	-1	1	1	-1	-1	1	1185	S_{qi}	9.02
1	1	-1	1	-1	1	-1	-1	3392		
1	-1	1	1	-1	-1	1	-1	1423		
1	1	1	1	1	1	1	1	3020		
20734.09	8779.754	1810.197	-2695.42	-793.583	-1172.46	-2077.44	-427.897	*Totals*		

Table 29. TPC-W Transparent Query Rewrite – 100,000 Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1651	SST	48745669.72
1	1	-1	-1	-1	-1	1	1	4218	SSY	333291238.00
1	-1	1	-1	-1	1	-1	1	3021	SS0	284545568.28
1	1	1	-1	1	-1	-1	-1	5575	S_e	46.30
1	-1	-1	1	1	-1	-1	1	1556	S_{qi}	9.45
1	1	-1	1	-1	1	-1	-1	4638		
1	-1	1	1	-1	-1	1	-1	2279		
1	1	1	1	1	1	1	1	4609		
27546.11	10532.91	3420.798	-1382.31	-765.214	289.4847	-2031.74	-738.089	*Totals*		

Table 30. TPC-W Transparent Query Rewrite – 100,000 Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	2349	SST	71244520.41
1	1	-1	-1	-1	-1	1	1	5881	SSY	582817240.00
1	-1	1	-1	-1	1	-1	1	4285	SS0	511572719.59
1	1	1	-1	1	-1	-1	-1	6891	S_e	33.16
1	-1	-1	1	1	-1	-1	1	2274	S_{qi}	6.77
1	1	-1	1	-1	1	-1	-1	5925		
1	-1	1	1	-1	-1	1	-1	3163		
1	1	1	1	1	1	1	1	6167		
36935	12794.1	4076.639	-1876.6	-1572.5	518.0404	-1814	277.8694	*Totals*		

Table 31. TPC-W Transparent Query Rewrite – 100,000 Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	2580	SST	136272394.03
1	1	-1	-1	-1	-1	1	1	7029	SSY	917010195.00
1	-1	1	-1	-1	1	-1	1	4986	SS0	780737800.97
1	1	1	-1	1	-1	-1	-1	9454	S_e	57.49
1	-1	-1	1	1	-1	-1	1	2694	S_{qi}	11.74
1	1	-1	1	-1	1	-1	-1	7382		
1	-1	1	1	-1	-1	1	-1	3824		
1	1	1	1	1	1	1	1	7679		
45628.58	17460.05	6259.298	-2469.57	-814.605	-375.081	-3404.53	-852.16	*Totals*		

Appendix N

TPC-W Benchmark Results for One Million Items

Table 32. TPC-W Authorization View – One Million Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1629	SST	1563979
1	1	-1	-1	-1	-1	1	1	1784	SSY	80422008
1	-1	1	-1	-1	1	-1	1	2051	SS0	78858029
1	1	1	-1	1	-1	-1	-1	2235	S_e	41.92
1	-1	-1	1	1	-1	-1	1	1394	S_{qi}	8.558
1	1	-1	1	-1	1	-1	-1	1860		
1	-1	1	1	-1	-1	1	-1	1599		
1	1	1	1	1	1	1	1	1950		
14501.31	1157.018	1168.305	-895.147	-86.4755	478.6679	-577.175	-144.816	*Totals*		

Table 33. TPC-W Authorization View – One Million Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3082	SST	5160174
1	1	-1	-1	-1	-1	1	1	3093	SSY	262520791
1	-1	1	-1	-1	1	-1	1	3963	SS0	257360617
1	1	1	-1	1	-1	-1	-1	4138	S_e	50.59
1	-1	-1	1	1	-1	-1	1	2774	S_{qi}	10.326
1	1	-1	1	-1	1	-1	-1	2968		
1	-1	1	1	-1	-1	1	-1	3045		
1	1	1	1	1	1	1	1	3135		
26197.23	470.4182	2362.821	-2355.03	59.9126	99.04876	-1488.38	-268.765	*Totals*		

Table 34. TPC-W Authorization View – One Million Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3807	SST	18540123
1	1	-1	-1	-1	-1	1	1	4236	SSY	546341469
1	-1	1	-1	-1	1	-1	1	5901	SS0	527801346
1	1	1	-1	1	-1	-1	-1	5896	S_e	68.65
1	-1	-1	1	1	-1	-1	1	3492	S_{qi}	14.014
1	1	-1	1	-1	1	-1	-1	4228		
1	-1	1	1	-1	-1	1	-1	4513		
1	1	1	1	1	1	1	1	5444		
37516.27	2091.312	5991.098	-2162.66	-238.583	1241.933	-1516.75	628.826	*Totals*		

Table 35. TPC-W Authorization View – One Million Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	4912	SST	29550838
1	1	-1	-1	-1	-1	1	1	5571	SSY	943321240
1	-1	1	-1	-1	1	-1	1	7577	SS0	913770402
1	1	1	-1	1	-1	-1	-1	8198	S_e	60.43
1	-1	-1	1	1	-1	-1	1	4964	S_{qi}	12.336
1	1	-1	1	-1	1	-1	-1	5749		
1	-1	1	1	-1	-1	1	-1	5913		
1	1	1	1	1	1	1	1	6479		
49363.16	2631.031	6972.907	-3153.17	-256.774	71.94694	-3613.32	-181.649	*Totals*		

Table 36. TPC-W Authorization View – One Million Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	6231	SST	35092403
1	1	-1	-1	-1	-1	1	1	6706	SSY	1473006520
1	-1	1	-1	-1	1	-1	1	9579	SS0	1437914117
1	1	1	-1	1	-1	-1	-1	9600	S_e	52.96
1	-1	-1	1	1	-1	-1	1	6868	S_{qi}	10.810
1	1	-1	1	-1	1	-1	-1	6962		
1	-1	1	1	-1	-1	1	-1	8037		
1	1	1	1	1	1	1	1	7941		
61922.84	493.5343	8389.967	-2309.65	-645.18	-497.733	-4093.45	264.8428	*Totals*		

Table 37. TPC-W Hippocratic Database – One Million Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1629	SST	1886683
1	1	-1	-1	-1	-1	1	1	1731	SSY	76657476
1	-1	1	-1	-1	1	-1	1	2051	SS0	74770793
1	1	1	-1	1	-1	-1	-1	2341	S_e	43.52
1	-1	-1	1	1	-1	-1	1	1394	S_{qi}	8.884
1	1	-1	1	-1	1	-1	-1	1752		
1	-1	1	1	-1	-1	1	-1	1599		
1	1	1	1	1	1	1	1	1624		
14120.51	776.2144	1109.939	-1383.32	-144.842	-9.50224	-953.659	-521.3	*Totals*		

Table 38. TPC-W Hippocratic Database – One Million Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3082	SST	5555757
1	1	-1	-1	-1	-1	1	1	3115	SSY	264937966
1	-1	1	-1	-1	1	-1	1	3963	SS0	259382209
1	1	1	-1	1	-1	-1	-1	4219	S_e	41.48
1	-1	-1	1	1	-1	-1	1	2774	S_{qi}	8.467
1	1	-1	1	-1	1	-1	-1	2970		
1	-1	1	1	-1	-1	1	-1	3045		
1	1	1	1	1	1	1	1	3133		
26299.92	573.1078	2417.462	-2457.75	114.5539	-3.67162	-1551.08	-331.462	*Totals*		

Table 39. TPC-W Hippocratic Database – One Million Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3807	SST	17456579
1	1	-1	-1	-1	-1	1	1	4249	SSY	532521434
1	-1	1	-1	-1	1	-1	1	5901	SS0	515064855
1	1	1	-1	1	-1	-1	-1	6031	S_e	60.67
1	-1	-1	1	1	-1	-1	1	3492	S_{qi}	12.385
1	1	-1	1	-1	1	-1	-1	4345		
1	-1	1	1	-1	-1	1	-1	4513		
1	1	1	1	1	1	1	1	4723		
37060.85	1635.892	5274.966	-2914.27	-954.715	490.3283	-2477.39	-331.812	*Totals*		

Table 40. TPC-W Hippocratic Database – One Million Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	4912	SST	27862284
1	1	-1	-1	-1	-1	1	1	5549	SSY	932684574
1	-1	1	-1	-1	1	-1	1	7577	SS0	904822290
1	1	1	-1	1	-1	-1	-1	8057	S_e	46.37
1	-1	-1	1	1	-1	-1	1	4964	S_{qi}	9.465
1	1	-1	1	-1	1	-1	-1	5732		
1	-1	1	1	-1	-1	1	-1	5913		
1	1	1	1	1	1	1	1	6417		
49120.87	2388.741	6807.192	-3069.43	-422.489	155.6931	-3539.49	-107.822	*Totals*		

Table 41. TPC-W Hippocratic Database – One Million Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	6231	SST	39201059
1	1	-1	-1	-1	-1	1	1	6754	SSY	1509066531
1	-1	1	-1	-1	1	-1	1	9579	SS0	1469865472
1	1	1	-1	1	-1	-1	-1	10057	S_e	59.23
1	-1	-1	1	1	-1	-1	1	6868	S_{qi}	12.091
1	1	-1	1	-1	1	-1	-1	7242		
1	-1	1	1	-1	-1	1	-1	8037		
1	1	1	1	1	1	1	1	7840		
62607.04	1177.736	8419.024	-2634.77	-616.122	-822.85	-4883.16	-524.866	*Totals*		

Table 42. TPC-W Label Based Access Control – One Million Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1629	SST	15139957
1	1	-1	-1	-1	-1	1	1	2857	SSY	153580603
1	-1	1	-1	-1	1	-1	1	2051	SS0	138440646
1	1	1	-1	1	-1	-1	-1	3695	S_e	39.09
1	-1	-1	1	1	-1	-1	1	1394	S_{qi}	7.978
1	1	-1	1	-1	1	-1	-1	2803		
1	-1	1	1	-1	-1	1	-1	1599		
1	1	1	1	1	1	1	1	3186		
19213.93	5869.634	1848.702	-1249.52	593.9212	124.2952	-671.58	-239.221	*Totals*		

Table 43. . TPC-W Label Based Access Control – One Million Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3082	SST	48630622
1	1	-1	-1	-1	-1	1	1	5167	SSY	440133427
1	-1	1	-1	-1	1	-1	1	3963	SS0	391502805
1	1	1	-1	1	-1	-1	-1	7273	S_e	59.57
1	-1	-1	1	1	-1	-1	1	2774	S_{qi}	12.159
1	1	-1	1	-1	1	-1	-1	3068		
1	-1	1	1	-1	-1	1	-1	3045		
1	1	1	1	1	1	1	1	3940		
32311.1	6584.291	4129.036	-6658.2	1826.128	-4204.13	-1843.43	-623.813	*Totals*		

Table 44. TPC-W Label Based Access Control – One Million Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3807	SST	142845245
1	1	-1	-1	-1	-1	1	1	7167	SSY	1149612762
1	-1	1	-1	-1	1	-1	1	5901	SS0	1006767517
1	1	1	-1	1	-1	-1	-1	10683	S_e	64.66
1	-1	-1	1	1	-1	-1	1	3492	S_{qi}	13.199
1	1	-1	1	-1	1	-1	-1	6755		
1	-1	1	1	-1	-1	1	-1	4513		
1	1	1	1	1	1	1	1	9497		
51814.22	16389.27	9372.868	-3300.46	3143.187	104.1331	-1846.84	298.7365	*Totals*		

Table 45. TPC-W Label Based Access Control – One Million Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	4912	SST	303396900
1	1	-1	-1	-1	-1	1	1	9847	SSY	2132259835
1	-1	1	-1	-1	1	-1	1	7577	SS0	1828862935
1	1	1	-1	1	-1	-1	-1	15649	S_e	44.21
1	-1	-1	1	1	-1	-1	1	4964	S_{qi}	9.024
1	1	-1	1	-1	1	-1	-1	8511		
1	-1	1	1	-1	-1	1	-1	5913		
1	1	1	1	1	1	1	1	12462		
69835.29	23103.17	13367.58	-6135.23	6137.897	-2910.11	-3567.15	-135.485	*Totals*		

Table 46. TPC-W Label Based Access Control – One Million Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	6231	SST	348819759
1	1	-1	-1	-1	-1	1	1	11088	SSY	3197112903
1	-1	1	-1	-1	1	-1	1	9579	SS0	2848293144
1	1	1	-1	1	-1	-1	-1	17612	S_e	57.59
1	-1	-1	1	1	-1	-1	1	6868	S_{qi}	11.755
1	1	-1	1	-1	1	-1	-1	12205		
1	-1	1	1	-1	-1	1	-1	8037		
1	1	1	1	1	1	1	1	15533		
87151.87	25722.57	14370.2	-1868.54	5335.051	-56.618	-5375.76	-1017.47	*Totals*		

Table 47. TPC-W Transparent Query Rewrite – One Million Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1629	SST	1658357
1	1	-1	-1	-1	-1	1	1	1967	SSY	81068792
1	-1	1	-1	-1	1	-1	1	2051	SS0	79410435
1	1	1	-1	1	-1	-1	-1	2236	S_e	51.46
1	-1	-1	1	1	-1	-1	1	1394	S_{qi}	10.505
1	1	-1	1	-1	1	-1	-1	1945		
1	-1	1	1	-1	-1	1	-1	1599		
1	1	1	1	1	1	1	1	1732		
14552.02	1207.721	682.8384	-1213.66	-571.942	160.1511	-698.473	-266.114	*Totals*		

Table 48. TPC-W Transparent Query Rewrite – One Million Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3082	SST	5329042
1	1	-1	-1	-1	-1	1	1	3400	SSY	274063588
1	-1	1	-1	-1	1	-1	1	3963	SS0	268734546
1	1	1	-1	1	-1	-1	-1	4242	S_e	43.33
1	-1	-1	1	1	-1	-1	1	2774	S_{qi}	8.844
1	1	-1	1	-1	1	-1	-1	3116		
1	-1	1	1	-1	-1	1	-1	3045		
1	1	1	1	1	1	1	1	3149		
26769.86	1043.047	2026.111	-2605.05	-276.797	-150.975	-1418.56	-198.95	*Totals*		

Table 49. TPC-W Transparent Query Rewrite – One Million Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3807	SST	16839544
1	1	-1	-1	-1	-1	1	1	4522	SSY	561100406
1	-1	1	-1	-1	1	-1	1	5901	SS0	544260862
1	1	1	-1	1	-1	-1	-1	6019	S_e	64.97
1	-1	-1	1	1	-1	-1	1	3492	S_{qi}	13.263
1	1	-1	1	-1	1	-1	-1	4827		
1	-1	1	1	-1	-1	1	-1	4513		
1	1	1	1	1	1	1	1	5017		
38096.75	2671.795	4801.995	-2400.15	-1427.69	1004.445	-2380.08	-234.501	*Totals*		

Table 50. TPC-W Transparent Query Rewrite – One Million Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	4912	SST	29827959
1	1	-1	-1	-1	-1	1	1	6125	SSY	967974963
1	-1	1	-1	-1	1	-1	1	7577	SS0	938147004
1	1	1	-1	1	-1	-1	-1	8371	S_e	43.60
1	-1	-1	1	1	-1	-1	1	4964	S_{qi}	8.901
1	1	-1	1	-1	1	-1	-1	5939		
1	-1	1	1	-1	-1	1	-1	5913		
1	1	1	1	1	1	1	1	6217		
50017.25	3285.126	6139.513	-3952.57	-1090.17	-727.45	-3683.8	-252.135	*Totals*		

Table 51. TPC-W Transparent Query Rewrite – One Million Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	6231	SST	35370069
1	1	-1	-1	-1	-1	1	1	7377	SSY	1563181275
1	-1	1	-1	-1	1	-1	1	9579	SS0	1527811206
1	1	1	-1	1	-1	-1	-1	10065	S_e	53.57
1	-1	-1	1	1	-1	-1	1	6868	S_{qi}	10.935
1	1	-1	1	-1	1	-1	-1	7548		
1	-1	1	1	-1	-1	1	-1	8037		
1	1	1	1	1	1	1	1	8125		
63829.17	2399.87	7782.279	-2674.64	-1252.87	-862.719	-4289.77	68.52138	*Totals*		

Appendix O

TPC-W Benchmark Results for 10 Million Items

Table 52. TPC-W Authorization View – 10 Million Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1957	SST	23691323.40
1	1	-1	-1	-1	-1	1	1	2477	SSY	127591594.00
1	-1	1	-1	-1	1	-1	1	3149	SS0	103900270.60
1	1	1	-1	1	-1	-1	-1	3676	S_e	46.01
1	-1	-1	1	1	-1	-1	1	533	S_{qi}	9.39
1	1	-1	1	-1	1	-1	-1	2068		
1	-1	1	1	-1	-1	1	-1	808		
1	1	1	1	1	1	1	1	1977		
16645.34	3750.867	2574.525	-5874.01	-360.33	1657.485	-2206.42	-374.159	*Totals*		

Table 53. TPC-W Authorization View – 10 Million Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3570	SST	-24690177.75
1	1	-1	-1	-1	-1	1	1	3805	SSY	373114808.53
1	-1	1	-1	-1	1	-1	1	5558	SS0	397804986.28
1	1	1	-1	1	-1	-1	-1	5512	S_e	40.52
1	-1	-1	1	1	-1	-1	1	3264	S_{qi}	8.27
1	1	-1	1	-1	1	-1	-1	3449		
1	-1	1	1	-1	-1	1	-1	3649		
1	1	1	1	1	1	1	1	3763		
32570.13	487.5022	4394.219	-4320.84	-350.54	109.3939	-2995.91	210.0333	*Totals*		

Table 54. TPC-W Authorization View – 10 Million Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	4988	SST	34884966.46
1	1	-1	-1	-1	-1	1	1	5408	SSY	880997765.00
1	-1	1	-1	-1	1	-1	1	7829	SS0	846112798.54
1	1	1	-1	1	-1	-1	-1	8127	S_e	43.43
1	-1	-1	1	1	-1	-1	1	4886	S_{qi}	8.87
1	1	-1	1	-1	1	-1	-1	5315		
1	-1	1	1	-1	-1	1	-1	5210		
1	1	1	1	1	1	1	1	5738		
47500.53	1675.533	6307.201	-5203.66	-22.3987	240.2942	-4813.87	219.5486	*Totals*		

Table 55. TPC-W Authorization View – 10 Million Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	6660	SST	53780380.66
1	1	-1	-1	-1	-1	1	1	7173	SSY	1486221413.00
1	-1	1	-1	-1	1	-1	1	10337	SS0	1432441032.34
1	1	1	-1	1	-1	-1	-1	10051	S_e	65.58
1	-1	-1	1	1	-1	-1	1	5965	S_{qi}	13.39
1	1	-1	1	-1	1	-1	-1	7179		
1	-1	1	1	-1	-1	1	-1	6819		
1	1	1	1	1	1	1	1	7621		
61804.88	2243.515	7851.309	-6638.31	-1210.75	1790.667	-5259.8	386.3932	*Totals*		

Table 56. TPC-W Authorization View – 10 Million Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	7973	SST	76514326.77
1	1	-1	-1	-1	-1	1	1	8535	SSY	2176150235.00
1	-1	1	-1	-1	1	-1	1	12216	SS0	2099635908.23
1	1	1	-1	1	-1	-1	-1	12453	S_e	47.70
1	-1	-1	1	1	-1	-1	1	7648	S_{qi}	9.74
1	1	-1	1	-1	1	-1	-1	8158		
1	-1	1	1	-1	-1	1	-1	8469		
1	1	1	1	1	1	1	1	9375		
74826.66	2214.349	10198.88	-7525.87	71.51849	617.6432	-6121.82	720.2128	*Totals*		

Table 57. TPC-W Hippocratic Database – 10 Million Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1957	SST	22559551.83
1	1	-1	-1	-1	-1	1	1	2344	SSY	128008921.00
1	-1	1	-1	-1	1	-1	1	3149	SS0	105449369.17
1	1	1	-1	1	-1	-1	-1	3547	S_e	46.15
1	-1	-1	1	1	-1	-1	1	533	S_{qi}	9.42
1	1	-1	1	-1	1	-1	-1	2419		
1	-1	1	1	-1	-1	1	-1	808		
1	1	1	1	1	1	1	1	2012		
16768.97	3874.495	2262.92	-5225.58	-671.936	2305.908	-2525.7	-693.432	*Totals*		

Table 58. TPC-W Hippocratic Database – 10 Million Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3570	SST	-24361922.32
1	1	-1	-1	-1	-1	1	1	4527	SSY	413982279.00
1	-1	1	-1	-1	1	-1	1	5558	SS0	438344201.32
1	1	1	-1	1	-1	-1	-1	5938	S_e	40.68
1	-1	-1	1	1	-1	-1	1	3264	S_{qi}	8.30
1	1	-1	1	-1	1	-1	-1	3824		
1	-1	1	1	-1	-1	1	-1	3649		
1	1	1	1	1	1	1	1	3859		
34189.44	2106.814	3818.856	-4996.84	-925.902	-566.61	-2979.89	226.0518	*Totals*		

Table 59. TPC-W Hippocratic Database – 10 Million Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	4988	SST	34797521.68
1	1	-1	-1	-1	-1	1	1	5446	SSY	881887171.00
1	-1	1	-1	-1	1	-1	1	7829	SS0	847089649.32
1	1	1	-1	1	-1	-1	-1	8144	S_e	41.30
1	-1	-1	1	1	-1	-1	1	4886	S_{qi}	8.43
1	1	-1	1	-1	1	-1	-1	5458		
1	-1	1	1	-1	-1	1	-1	5210		
1	1	1	1	1	1	1	1	5569		
47527.95	1702.945	5973.752	-5284.95	-355.847	159.0077	-5103.66	-70.2384	*Totals*		

Table 60. TPC-W Hippocratic Database – 10 Million Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	6660	SST	66389767.01
1	1	-1	-1	-1	-1	1	1	7009	SSY	1511931194.00
1	-1	1	-1	-1	1	-1	1	10337	SS0	1445541426.99
1	1	1	-1	1	-1	-1	-1	10753	S_e	50.02
1	-1	-1	1	1	-1	-1	1	5965	S_{qi}	10.21
1	1	-1	1	-1	1	-1	-1	6994		
1	-1	1	1	-1	-1	1	-1	6819		
1	1	1	1	1	1	1	1	7550		
62086.85	2525.49	8830.675	-7432.92	-231.386	996.0552	-6012.14	-365.938	*Totals*		

Table 61. TPC-W Hippocratic Database – 10 Million Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	7973	SST	79259146.15
1	1	-1	-1	-1	-1	1	1	8337	SSY	2189738354.00
1	-1	1	-1	-1	1	-1	1	12216	SS0	2110479207.85
1	1	1	-1	1	-1	-1	-1	12659	S_e	49.32
1	-1	-1	1	1	-1	-1	1	7648	S_{qi}	10.07
1	1	-1	1	-1	1	-1	-1	8561		
1	-1	1	1	-1	-1	1	-1	8469		
1	1	1	1	1	1	1	1	9157		
75019.63	2407.317	9980.7	-7350.07	-146.663	793.4394	-7146.81	-304.782	*Totals*		

Table 62. TPC-W Label Based Access Control – 10 Million Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1957	SST	51329835.91
1	1	-1	-1	-1	-1	1	1	3704	SSY	241429294.00
1	-1	1	-1	-1	1	-1	1	3149	SS0	190099458.09
1	1	1	-1	1	-1	-1	-1	4957	S_e	54.46
1	-1	-1	1	1	-1	-1	1	533	S_{qi}	11.12
1	1	-1	1	-1	1	-1	-1	3540		
1	-1	1	1	-1	-1	1	-1	808		
1	1	1	1	1	1	1	1	3867		
22515.15	9620.674	3046.435	-5020.09	111.5797	2511.397	-1842.24	-9.97863	*Totals*		

Table 63. TPC-W Label Based Access Control – 10 Million Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3570	SST	-81161470.44
1	1	-1	-1	-1	-1	1	1	6371	SSY	693276063.00
1	-1	1	-1	-1	1	-1	1	5558	SS0	774437533.44
1	1	1	-1	1	-1	-1	-1	9344	S_e	47.41
1	-1	-1	1	1	-1	-1	1	3264	S_{qi}	9.68
1	1	-1	1	-1	1	-1	-1	5966		
1	-1	1	1	-1	-1	1	-1	3649		
1	1	1	1	1	1	1	1	7723		
45444.11	13361.48	7102.525	-4240.89	2357.767	189.342	-2818.75	387.1921	*Totals*		

Table 64.TPC-W Label Based Access Control – 10 Million Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	4988	SST	234640734.28
1	1	-1	-1	-1	-1	1	1	8671	SSY	1906151808.00
1	-1	1	-1	-1	1	-1	1	7829	SS0	1671511073.72
1	1	1	-1	1	-1	-1	-1	12929	S_e	55.93
1	-1	-1	1	1	-1	-1	1	4886	S_{qi}	11.42
1	1	-1	1	-1	1	-1	-1	9090		
1	-1	1	1	-1	-1	1	-1	5210		
1	1	1	1	1	1	1	1	13161		
66763.48	20938.48	11493.06	-2070.85	5163.458	3373.111	-2703.14	2330.28	*Totals*		

Table 65. TPC-W Label Based Access Control – 10 Million Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	6660	SST	392633865.42
1	1	-1	-1	-1	-1	1	1	11567	SSY	3124982748.00
1	-1	1	-1	-1	1	-1	1	10337	SS0	2732348882.58
1	1	1	-1	1	-1	-1	-1	18362	S_e	46.40
1	-1	-1	1	1	-1	-1	1	5965	S_{qi}	9.47
1	1	-1	1	-1	1	-1	-1	10770		
1	-1	1	1	-1	-1	1	-1	6819		
1	1	1	1	1	1	1	1	14879		
85359.61	25798.25	15434.76	-8494.85	6372.695	-65.8682	-5509.4	136.7946	*Totals*		

Table 66. TPC-W Label Based Access Control – 10 Million Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	7973	SST	583845493.41
1	1	-1	-1	-1	-1	1	1	13728	SSY	4646462431.00
1	-1	1	-1	-1	1	-1	1	12216	SS0	4062616937.59
1	1	1	-1	1	-1	-1	-1	22254	S_e	37.53
1	-1	-1	1	1	-1	-1	1	7648	S_{qi}	7.66
1	1	-1	1	-1	1	-1	-1	13018		
1	-1	1	1	-1	-1	1	-1	8469		
1	1	1	1	1	1	1	1	18779		
104084.8	31472.49	19349.68	-8256.44	9222.321	-112.927	-6186.35	655.6839	*Totals*		

Table 67. TPC-W Label Based Access Control – 10 Million Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3570	SST	-33661289.01
1	1	-1	-1	-1	-1	1	1	3873	SSY	378872768.00
1	-1	1	-1	-1	1	-1	1	5558	SS0	412534057.01
1	1	1	-1	1	-1	-1	-1	5439	S_e	45.52
1	-1	-1	1	1	-1	-1	1	3264	S_{qi}	9.29
1	1	-1	1	-1	1	-1	-1	3754		
1	-1	1	1	-1	-1	1	-1	3649		
1	1	1	1	1	1	1	1	4060		
33167.62	1084.99	4243.791	-3712.05	-500.968	718.1785	-2862.94	343.0022	*Totals*		

Table 68. TPC-W Label Based Access Control – 10 Million Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	4988	SST	30903987.63
1	1	-1	-1	-1	-1	1	1	5532	SSY	901634461.00
1	-1	1	-1	-1	1	-1	1	7829	SS0	870730473.37
1	1	1	-1	1	-1	-1	-1	7942	S_e	42.57
1	-1	-1	1	1	-1	-1	1	4886	S_{qi}	8.69
1	1	-1	1	-1	1	-1	-1	5744		
1	-1	1	1	-1	-1	1	-1	5210		
1	1	1	1	1	1	1	1	6057		
48186.59	2361.593	5888.46	-4395.57	-441.139	1048.389	-4614.31	419.1125	*Totals*		

Table 69. TPC-W Label Based Access Control – 10 Million Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	6660	SST	392633865.42
1	1	-1	-1	-1	-1	1	1	11567	SSY	3124982748.00
1	-1	1	-1	-1	1	-1	1	10337	SS0	2732348882.58
1	1	1	-1	1	-1	-1	-1	18362	S_e	46.40
1	-1	-1	1	1	-1	-1	1	5965	S_{qi}	9.47
1	1	-1	1	-1	1	-1	-1	10770		
1	-1	1	1	-1	-1	1	-1	6819		
1	1	1	1	1	1	1	1	14879		
85359.61	25798.25	15434.76	-8494.85	6372.695	-65.8682	-5509.4	136.7946	*Totals*		

Table 70. TPC-W Label Based Access Control – 10 Million Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	7973	SST	583845493.41
1	1	-1	-1	-1	-1	1	1	13728	SSY	4646462431.00
1	-1	1	-1	-1	1	-1	1	12216	SS0	4062616937.59
1	1	1	-1	1	-1	-1	-1	22254	S_e	37.53
1	-1	-1	1	1	-1	-1	1	7648	S_{qi}	7.66
1	1	-1	1	-1	1	-1	-1	13018		
1	-1	1	1	-1	-1	1	-1	8469		
1	1	1	1	1	1	1	1	18779		
104084.8	31472.49	19349.68	-8256.44	9222.321	-112.927	-6186.35	655.6839	*Totals*		

Table 71. TPC-W Transparent Query Rewrite – 10 Million Items – 800 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	1957	SST	20682600.09
1	1	-1	-1	-1	-1	1	1	2240	SSY	125073825.00
1	-1	1	-1	-1	1	-1	1	3149	SS0	104391224.91
1	1	1	-1	1	-1	-1	-1	3289	S_e	52.76
1	-1	-1	1	1	-1	-1	1	533	S_{qi}	10.77
1	1	-1	1	-1	1	-1	-1	2544		
1	-1	1	1	-1	-1	1	-1	808		
1	1	1	1	1	1	1	1	2165		
16684.62	3790.148	2136.979	-4586.22	-797.876	2945.266	-2344.36	-512.097	*Totals*		

Table 72. TPC-W Transparent Query Rewrite – 10 Million Items – 1600 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	3570	SST	-33661289.01
1	1	-1	-1	-1	-1	1	1	3873	SSY	378872768.00
1	-1	1	-1	-1	1	-1	1	5558	SS0	412534057.01
1	1	1	-1	1	-1	-1	-1	5439	S_e	45.52
1	-1	-1	1	1	-1	-1	1	3264	S_{qi}	9.29
1	1	-1	1	-1	1	-1	-1	3754		
1	-1	1	1	-1	-1	1	-1	3649		
1	1	1	1	1	1	1	1	4060		
33167.62	1084.99	4243.791	-3712.05	-500.968	718.1785	-2862.94	343.0022	*Totals*		

Table 73. TPC-W Transparent Query Rewrite – 10 Million Items – 2400 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	4988	SST	30903987.63
1	1	-1	-1	-1	-1	1	1	5532	SSY	901634461.00
1	-1	1	-1	-1	1	-1	1	7829	SS0	870730473.37
1	1	1	-1	1	-1	-1	-1	7942	S_e	42.57
1	-1	-1	1	1	-1	-1	1	4886	S_{qi}	8.69
1	1	-1	1	-1	1	-1	-1	5744		
1	-1	1	1	-1	-1	1	-1	5210		
1	1	1	1	1	1	1	1	6057		
48186.59	2361.593	5888.46	-4395.57	-441.139	1048.389	-4614.31	419.1125	*Totals*		

Table 74. TPC-W Transparent Query Rewrite – 10 Million Items – 3200 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	6660	SST	60621905.08
1	1	-1	-1	-1	-1	1	1	7254	SSY	1494730394.00
1	-1	1	-1	-1	1	-1	1	10337	SS0	1434108488.92
1	1	1	-1	1	-1	-1	-1	10427	S_e	50.57
1	-1	-1	1	1	-1	-1	1	5965	S_{qi}	10.32
1	1	-1	1	-1	1	-1	-1	6857		
1	-1	1	1	-1	-1	1	-1	6819		
1	1	1	1	1	1	1	1	7522		
61840.84	2279.477	8368.719	-7516.24	-693.342	912.7381	-5330.12	316.0727	*Totals*		

Table 75. TPC-W Transparent Query Rewrite – 10 Million Items – 4000 EBS's

I	A	B	C	AB	AC	BC	ABC	Mean ÿ	Variation	Sum of Squares
1	-1	-1	-1	1	1	1	-1	7973	SST	74743136.89
1	1	-1	-1	-1	-1	1	1	8849	SSY	2198354531.00
1	-1	1	-1	-1	1	-1	1	12216	SS0	2123611394.11
1	1	1	-1	1	-1	-1	-1	12461	S_e	47.45
1	-1	-1	1	1	-1	-1	1	7648	S_{qi}	9.69
1	1	-1	1	-1	1	-1	-1	8246		
1	-1	1	1	-1	-1	1	-1	8469		
1	1	1	1	1	1	1	1	9391		
75252.67	2640.355	9821.083	-7743.69	-306.281	399.8164	-5887.66	954.368	*Totals*		

Appendix P

Wildlife Data Benchmark Results

Table 76. Wildlife Data Authorization Views – 800 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	1562	1544	1588	1565	SST	8833340320
1	1	-1	-1	1663	1625	1672	1653	SSY	18743828157
1	-1	1	-1	55363	55385	55386	55378	SS0	9910487837
1	1	1	1	56313	56386	56370	56356	S_e	37.11
114952.09	1066.97	108516.55	889.68				Totals	S_{qi}	10.71

Table 77. Wildlife Data Authorization Views – 1600 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	2985	2990	2927	2967	SST	35383610236
1	1	-1	-1	3287	3275	3240	3267	SSY	74940625248
1	-1	1	-1	110881	110588	110900	110790	SS0	39557015011
1	1	1	1	112574	112537	112790	112634	S_e	159.86
229657.76	2144.22	217188.61	1543.85				Totals	S_{qi}	46.15

Table 78. Wildlife Data Authorization Views – 2400 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	4016	4036	4023	4025	SST	79776359342
1	1	-1	-1	4845	4869	4817	4844	SSY	168454645778
1	-1	1	-1	166532	166123	166076	166244	SS0	88678286436
1	1	1	1	168628	168897	168710	168745	S_e	203.32
343857.11	3320.05	326119.89	1682.79				Totals	S_{qi}	58.69

Table 79. Wildlife Data Authorization Views – 3200 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	5367	5362	5372	5367	SST	142396001745
1	1	-1	-1	5957	5907	5972	5945	SSY	299938324484
1	-1	1	-1	221561	221476	221625	221554	SS0	157542322739
1	1	1	1	225629	225609	225121	225453	S_e	211.61
458319.14	4477.16	435694.60	3320.62				Totals	S_{qi}	61.09

Table 80. Wildlife Data Authorization Views – 4000 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	6530	6568	6528	6542	SST	222950772667
1	1	-1	-1	7229	7210	7259	7233	SSY	468937015267
1	-1	1	-1	276221	276247	276411	276293	SS0	245986242600
1	1	1	1	282829	282286	282773	282629	S_e	224.65
572696.83	7026.91	545147.60	5645.58				Totals	S_{qi}	64.85

Table 81. Wildlife Data Hippocratic Database – 800 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	1353	1349	1346	1349	SST	6360993393
1	1	-1	-1	1453	1416	1461	1443	SSY	12919565900
1	-1	1	-1	35660	35681	35695	35679	SS0	6558572506
1	1	1	1	55012	55040	55076	55042	S_e	31.07
93513.44	19457.45	87928.38	19269.71				Totals	S_{qi}	8.97

Table 82. Wildlife Data Hippocratic Database – 1600 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	2917	2992	2915	2941	SST	25409782366
1	1	-1	-1	2887	2804	2806	2832	SSY	51680626161
1	-1	1	-1	71183	71150	71258	71197	SS0	26270843795
1	1	1	1	110268	110052	110241	110187	S_e	102.65
187157.13	38880.93	175610.76	39099.02				Totals	S_{qi}	29.63

Table 83. Wildlife Data Hippocratic Database – 2400 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	4082	3961	3998	4013	SST	57455635210
1	1	-1	-1	4168	4151	4105	4141	SSY	116435872752
1	-1	1	-1	106753	106973	106913	106880	SS0	58980237542
1	1	1	1	165515	165164	165505	165395	S_e	169.96
280428.81	58642.86	264119.62	58387.06				Totals	S_{qi}	49.06

Table 84. Wildlife Data Hippocratic Database – 3200 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	5664	5595	5582	5614	SST	102963337474
1	1	-1	-1	5404	5428	5421	5418	SSY	208868959816
1	-1	1	-1	143325	143317	143460	143367	SS0	105905622342
1	1	1	1	221392	221396	221344	221377	S_e	68.50
375775.86	77814.12	353713.44	78205.91				Totals	S_{qi}	19.77

Table 85. Wildlife Data Hippocratic Database – 4000 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	6792	6649	6751	6730	SST	159892343556
1	1	-1	-1	6937	6927	6972	6945	SSY	324333309640
1	-1	1	-1	178666	178789	178966	178807	SS0	164440966084
1	1	1	1	275817	275741	275733	275764	S_e	124.26
468246.33	97171.62	440894.92	96741.80				Totals	S_{qi}	35.87

Table 86. Wildlife Data Label Based Access Control – 800 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	1353	1349	1346	1349	SST	6421471065
1	1	-1	-1	2332	2355	2315	2334	SSY	13236841340
1	-1	1	-1	35660	35681	35695	35679	SS0	6815370275
1	1	1	1	55983	55932	55979	55965	S_e	27.66
95326.60	21270.61	87960.05	19301.39				Totals	S_{qi}	7.98

Table 87. Wildlife Data Label Based Access Control – 1600 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	2917	2992	2915	2941	SST	25572825288
1	1	-1	-1	4727	4781	4794	4767	SSY	52886612343
1	-1	1	-1	71183	71150	71258	71197	SS0	27313787055
1	1	1	1	111917	111969	111906	111931	S_e	60.74
190836.01	42559.81	175419.29	38907.55				Totals	S_{qi}	17.54

Table 88. Wildlife Data Label Based Access Control – 2400 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	4082	3961	3998	4013	SST	57791203335
1	1	-1	-1	6860	6890	6800	6850	SSY	118951651370
1	-1	1	-1	106753	106973	106913	106880	SS0	61160448035
1	1	1	1	167637	167861	167968	167822	S_e	153.96
285564.82	63778.87	263838.32	58105.77				Totals	S_{qi}	44.44

Table 89. Wildlife Data Label Based Access Control – 3200 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	5664	5595	5582	5614	SST	103969971218
1	1	-1	-1	9057	9062	9020	9046	SSY	214094653642
1	-1	1	-1	143325	143317	143460	143367	SS0	110124682424
1	1	1	1	225035	225293	225154	225161	S_e	113.13
383187.83	85226.09	353868.09	78360.56				Totals	S_{qi}	32.66

Table 90. Wildlife Data Label Based Access Control – 4000 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	6792	6649	6751	6730	SST	161781209961
1	1	-1	-1	11347	11358	11334	11346	SSY	332868278384
1	-1	1	-1	178666	178789	178966	178807	SS0	171087068423
1	1	1	1	280710	280634	280850	280731	S_e	141.99
477615.00	106540.29	441461.55	97308.43				Totals	S_{qi}	40.99

Table 91. Wildlife Data Transparent Query Rewrite – 800 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	1353	1349	1346	1349	SST	6362840864
1	1	-1	-1	1704	1795	1792	1763	SSY	12998794684
1	-1	1	-1	35660	35681	35695	35679	SS0	6635953820
1	1	1	1	55295	55261	55261	55272	S_e	41.20
94063.48	20007.49	87838.51	19179.84				Totals	S_{qi}	11.89

Table 92. Wildlife Data Transparent Query Rewrite – 1600 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	2917	2992	2915	2941	SST	25406693137
1	1	-1	-1	3290	3246	3272	3269	SSY	51883226030
1	-1	1	-1	71183	71150	71258	71197	SS0	26476532893
1	1	1	1	110519	110477	110447	110481	S_e	58.23
187888.38	39612.18	175467.58	38955.84				Totals	S_{qi}	16.81

Table 93. Wildlife Data Transparent Query Rewrite – 2400 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	4082	3961	3998	4013	SST	57298911268
1	1	-1	-1	4994	5004	4995	4998	SSY	116773513182
1	-1	1	-1	106753	106973	106913	106880	SS0	59474601914
1	1	1	1	165618	165835	165680	165711	S_e	121.03
281601.61	59815.66	263579.59	57847.03				Totals	S_{qi}	34.94

Table 94. Wildlife Data Transparent Query Rewrite – 3200 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	5664	5595	5582	5614	SST	102842970354
1	1	-1	-1	6214	6272	6285	6257	SSY	209461652709
1	-1	1	-1	143325	143317	143460	143367	SS0	106618682355
1	1	1	1	221892	221795	221716	221801	S_e	93.80
377038.78	79077.04	353297.83	77790.30				Totals	S_{qi}	27.08

Table 95. Wildlife Data Transparent Query Rewrite – 4000 EBS's

I	A	B	AB	y_{i1}	y_{i2}	y_{i3}	Mean ÿ	Variation	Sum of Squares
1	-1	-1	1	6792	6649	6751	6730	SST	160511853941
1	1	-1	-1	7685	7609	7693	7662	SSY	326263178624
1	-1	1	-1	178666	178789	178966	178807	SS0	165751324683
1	1	1	1	276944	276985	276797	276909	S_e	141.59
470108.25	99033.54	441322.98	97169.86				Totals	S_{qi}	40.87

Reference List

Agrawal, R., Ailamaki, A., Bernstein, P., A., Brewer, E., A. , Carey, M., J. , Chaudhuri, S., et al. (2009). The Claremont report on database research. *Communications of the ACM, 52*(6), 56-65.

Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2002). Hippocratic databases. In *Proceedings of the 28th International Conference on Very Large Data Bases* (pp. 143-154). San Francisco, CA: Morgan Kaufmann.

Agrawal, R., Bird, P., Grandison, T., Kiernan, J., Logan, S., & Rjaibi, W. (2005). Extending relational database systems to automatically enforce privacy policies. In *Proceedings of the 21st International Conference on Data Engineering* (pp. 1013-1022). Washington, DC: IEEE Computer Society.

Agrawal, R., Grandison, T., Johnson, C., & Kiernan, J. (2007). Enabling the 21st century health care information technology revolution. *Communications of the ACM 50*(2), 34-42.

Ahmad, M., Duan, S., Aboulnaga, A., & Babu, S. (2011). Predicting completion times of batch query workloads using interaction-aware models and simulation. In *Proceedings of the 14th International Conference on Extending Database Technology* (pp. 449-460). New York, NY: ACM.

Apache Software Foundation (n.d.). Jmeter user manual. Retrieved from http://jakarta.apache.org/jmeter/usermanual/index.html

Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., et al. (1976). System R: Relational approach to database management. *ACM Transactions on Database Systems 1(*2), 97-137.

Avritzer, A., & Weyuker, E., J. (2004). The role of modeling in the performance testing of e-commerce applications. *IEEE Transactions on Software Engineering, 30*(12), 1072-1083.

Aziz, B., Arenas, A., Martinelli, F., Matteucci, I., & Mori, P. (2008). Controlling usage in business process workflows through fine-grained security policies. In *Proceedings of the 5th International Conference on Trust, Privacy and Security in Digital Business* (pp. 100-117). Berlin: Springer-Verlag.

Bell, D. E. (2005). Looking back at the Bell-La Padula model. In *Proceedings of the 21st Annual Computer Security Applications Conference* (pp. 337-351). Washington, DC: IEEE Computer Society.

Benantar, M. (2005). *Access control systems: Security, identity management and trust models*. New York, NY: Springer-Verlag Inc.

Bertino, E., Byun, J.-W., & Li, N. (2005). Privacy-preserving database systems. *Lecture Notes in Computer Science, 3655*, 178-206.

Bertino, E., & Sandhu, R. (2005). Database security-concepts, approaches, and challenges. *IEEE Transactions on Dependable and Secure Computing, 2*(01), 2-19.

Bishop, M. A. (2002). *Computer security: Art and science*. New York: Addison-Wesley.

Bitton, D., DeWitt, D. J., & Turbyfill, C. (1983). Benchmarking database systems: A systematic approach. In M. Schkolnick & C. Thanos (Eds.), *Proceedings of the 9th International Conference on Very Large Data Bases* (pp. 8-19). San Francisco, CA: Morgan Kaufmann.

Blasgen, M. W., Astrahan, M. M., Chamberlin, D. D., Gray, J. N., King, W. F., Lindsay, B. G., et al. (1981). System R: An architectural overview. *IBM Systems Journal, 20*(1), 41-62.

Bond, R., See, K. Y.-K., Wong, C. K. M., & Chan, Y.-K. H. (2006). *Understanding DB2 9 security*. Indianapolis, IL: IBM Press.

Boote, D. N., & Beile, P. (2005). On the centrality of the dissertation literature review in research preparation. *Educational Researcher, 34*(6), 3-15.

Bruns, G., Dantas, D. S., & Huth, M. (2007). A simple and expressive semantic framework for policy composition in access control. In *Proceedings of the 2007 ACM Workshop on Formal Methods in Security Engineering* (pp. 12-21). New York, NY: ACM.

Castano, S., Fugini, M. G., Martella, G., & Samarati, P. (1994). *Database security*. New York, NY: ACM Press/Addison-Wesley.

cell. (n.d.). *The American Heritage dictionary of the English language, fourth edition*. Retrieved from Dictionary.com website: http://dictionary.reference.com/browse/cell

Chakraborty, A., Majumdar, A. K., & Sural, S. (2010). A column dependency-based approach for static and dynamic recovery of databases from malicious transactions. *International Journal of Information Security, 9*(1), 51-67.

Chamberlin, D. D., Astrahan, M. M., Blasgen, M. W., Gray, J. N., King, W. F., Lindsay, B. G., et al. (1981). A history and evaluation of System R. *Communications of the ACM, 24*(10), 632-646.

Chaudhuri, S., Dutta, T., & Sudarshan, S. (2007). Fine grained authorization through predicated grants. In *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering* (pp. 1174-1183). Washington, DC: IEEE Computer Society.

Chaudhuri, S. (2009). Query optimizers: Time to rethink the contract? In *Proceedings of the 35th SIGMOD International Conference on Management of Data* (961-968). New York, NY: ACM.

Cho, E., Kim, Y., Hong, M., & Cho, W. (2009). Fine-grained view-based access control for RDF cloaking. In *Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology* (pp. 336-341). Washington, DC: IEEE Computer Society.

Corcoran, B. J., Swamy, N., & Hicks, M. (2009). Cross-tier, label-based security enforcement for web applications. In *Proceedings of the 35th SIGMOD International Conference on Management of Data* (pp 269-282). New York, NY: ACM.

Cuppens, F., & Cuppens-Boulahia, N. (2008). Modeling contextual security policies. *International Journal of Information Security, 7*(4), 285-305.

Currim, F., Jung, E., Xiao, X., & Jo, I. (2009). Privacy policy enforcement for health information data access. In *Proceedings of the 1st ACM International Workshop on Medical-grade Wireless Networks* (pp. 39-44). New York, NY: ACM.

Dawson, C. W. (2000). *The essence of computing projects: A student's guide*. Harlow, Essex, UK: Pearson.

Dietrich, S. W. (2001). *Understanding relational database query languages*. Upper Saddle River, NJ: Prentice Hall.

Denning, D. E., Akl, S. G., Heckman, M., Lunt, T. F., Morgenstern, M., Neumann, P. G., et al. (1987). Views for multilevel database security. *IEEE Transactions on Software Engineering, 13*(2), 129-140.

Deshpande, A., Ives, Z., & Raman, V. (2007). Adaptive query processing. *Foundations and Trends in Databases, 1*(1), 1-140.

DeWitt, D. J. (1992). The Wisconsin benchmark: Past, present, and future. In J. Gray (Ed.), *The benchmark handbook for database and transaction systems* (pp. 269-315). San Mateo, CA: Morgan Kaufmann.

Dixit, K. M. (1992). Overview of the SPEC Benchmark. In J. Gray (Ed.), *The Benchmark Handbook* (pp. 489-524). San Mateo, CA: Morgan Kaufmann.

Elmasri, R. A., & Navathe, S. B. (2010). *Fundamentals of database systems* (6th ed.). Boston, MA: Addison-Wesley Longman.

Elnikety, S., Dropsho, S., Cecchet, E., Zwaenepoel, W. (2009). Predicting replicated database scalability from standalone database profiling. In *Proceedings of the Fourth ACM European Conference on Computer Systems* (pp. 303-316). New York, NY: ACM.

Fischer, J., Marino, D., Majumdar, R., & Millstein, T. (2009). Fine-grained access control with object-sensitive roles. In *Proceedings of the 23rd European Conference Object-Oriented Programming* (pp. 173-194). Berlin: Springer-Verlag.

Franzoni, S., Mazzoleni, P., Valtolina, S., & Bertino, E. (2007). Towards a fine-grained access control model and mechanisms for semantic databases. In *Proceedings of the IEEE International Conference on Web Services* (pp. 993-1000). Washington, DC: IEEE Computer Society.

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). *Database systems: The complete book* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Gonzalez, A. G. (2006). Open science: Open source licenses in scientific research. *North Carolina Journal of Law & Technology*, *7*(2), 321-366.

Goyal, V., Pandey, O., Sahai, A., & Waters, B. (2006). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (pp. 88-98), New York, NY: ACM.

Gray, J. (1992). *The benchmark handbook for database and transaction systems*. San Mateo, CA: Morgan Kaufmann.

Griffiths, P. P., & Wade, B. W. (1976). An authorization mechanism for a relational database system. *ACM Transactions on Database Systems 1*(3), 242-255.

Gunther, N. J. (2004, March 11). Benchmark blunders and things that go bump in the night. Retrieved from http://arxiv.org/PS_cache/cs/pdf/0404/0404043v1.pdf

Halevy, A. Y. (2001). Answering queries using views: A survey. *The VLDB Journal 10*(4), 270-294.

He, D. D., & Yang, J. (2009). Authorization control in collaborative healthcare systems. *Journal of Theoretical and Applied Electronic Commerce Research, 4*(2), 88.

He, Z., & Veeraraghavan, P. (2009). Fine-grained updates in database management systems for flash memory. *Information Sciences: an International Journal 179*(18), 3162-3181.

Hellerstein, J. M., Stonebraker, M., & Hamilton, J. (2007). Architecture of a database system. *Foundations and Trends in Databases 1*(2): 141-259.

Hill, M. D., & Marty, M. R. (2008). Amdahl's Law in the Multicore Era. *Computer, 41*(7), 33-38.

Howard, K., & Sharp, J. A. (1996). *The management of a student research project* (2nd ed.). Aldershot, UK: Gower.

Jain, R. (1991). *The art of computer systems performance analysis*. New York, NY: John Wiley & Sons.

Jha, S., Li, N., Tripunitara, M., Wang, Q., & Winsborough, W. (2008). Towards formal verification of role-based access control policies. *IEEE Transactions on Dependable and Secure Computing 5*(4), 242-255.

Jin, Y., Tang, A., Han, J., & Liu, Y. (2007). Performance evaluation and prediction for legacy information systems. In *Proceedings of the 29th International Conference on Software Engineering* (pp. 540-549). Washington, DC: IEEE Computer Society.

Johnson, C. M., & Grandison, T. W. A. (2007). Compliance with data protection laws using Hippocratic database active enforcement and auditing. *IBM Systems Journal, 46*(2), 255-264.

Kabra, G., Ramamurthy, R., & Sudarshan, S. (2006). Redundancy and information leakage in fine-grained access control. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data* (pp. 133-144). New York: ACM Press.

Kalibera, T., Bulej, L., & Tuma, P. (2005, July 28). *Benchmark precision and random initial state*. Retrieved from http://d3s.mff.cuni.cz/publications/KaliberaBulejTuma-BenchmarkPrecision.pdf

Kalibera, T., & Tuma, P. (2006). Precise regression benchmarking with random effects: Improving mono benchmark results. *Lecture Notes in Computer Science, 4054*, 63-77.

Kanza, Y., Mendelzon, A., Miller, R., & Zhang, Z. (2006). Authorization-transparent access control for XML under the Non-Truman model. In Y. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust & C. Boehm (Eds.), *Advances in Database Technology* (pp. 222-239). Berlin: Springer.

Keeton, K., & Patterson, D. A. (2000). Towards a simplified database workload for computer architecture evaluations. In L. K. John & A. M. Maynard (Eds.), *Workload Characterization for Computer System Design* (pp. 49-69). Norwell, MA: Kluwer Academic Publishers.

Kocaturk, M. M., & Gundem, T. J. (2008). A fine-grained access control system combining MAC and RBACK models for XML. *Informatica, 19*(4), 517-534.

Krishnamurthy, A., Mettler, A., & Wagner, D. (2010). Fine-grained privilege separation for web applications. In *Proceedings of the 19th International Conference on World Wide Web* (pp. 551-560). New York, NY: ACM.

LeFevre, K., Agrawal, R., Ercegovac, V., Ramakrishnan, R., Xu, Y., & DeWitt, D. (2004). Limiting disclosure in Hippocratic databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases* (pp. 108-119). Berlin: Springer.

Majedi, M., Ghazinour, K., Chinaei, A. H., & Barker, K. (2009). SQL privacy model for social networks. In *Proceeding of the 2009 International Conference on Advances in Social Network Analysis and Mining* (pp. 369-370). Washington, DC: IEEE Computer Society.

Manjhi, A., Ailamaki, A., Maggs, B. M., Mowry, T. C., Olston, C., & Tomasic, A. (2006). Simultaneous scalability and security for data-intensive web applications. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data* (pp. 241-252). New York, NY: ACM Press.

Matos, V. (1996). Evaluating the cost of protecting operational databases. *Journal of Systems Management, 47*(1), 36-45.

Mazzoleni, P., Crispo, B., Sivasubramanian, S., & Bertino, E. (2005). Efficient integration of fine-grained access control in large-scale grid services. In *Proceedings of the 2005 IEEE International Conference on Services Computing* (pp. 77-86). Washington, DC: IEEE Computer Society.

Melnik, S., Rahm, E., & Bernstein, P. A. (2003). Rondo: A programming platform for generic model management. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (pp. 193-204). New York, NY: ACM Press.

Menascé, D. A., & Almeida, V. (2001). *Capacity planning for web services: Metrics, models, and methods*. Upper Saddle River, NJ: Prentice Hall.

Menascé, D. A. (2002). TPC-W: A benchmark for e-commerce. *IEEE Internet Computing, 6*(3), 83-87.

Mi, N., Casale, G., Cherkasova, L., & Smirni, E. (2009). Injecting realistic burstiness to a traditional client-server benchmark. In *Proceedings of the 6th International Conference on Autonomic Computing* (pp. 149-158). New York, NY: ACM.

Michiels, P., Manolescu, I., & Miachon, C. (2008). Toward microbenchmarking XQuery. *Information Systems 33*(2), 182-202.

Minami, K. (2006). Secure context-sensitive authorization. *Dissertation Abstracts International, 67* (06), B. (UMI No. 3219711) Retrieved June 26, 2007, from Digital Dissertations database.

Moore, N. (2011). Computational complexity of the problem of tree generation under fine-grained access control policies. *Information and Computation 209*(3): 548-567.

Motro, A. (1989). An access authorization model for relational databases based on algebraic manipulation of view definitions. In *Proceedings of the Fifth International Conference on Data Engineering (*pp. 339-347). Washington, DC: IEEE Computer Society.

Ni, Q., Trombetta, A., Bertino, E., & Lobo, J. (2007). Privacy-aware role based access control. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies* (pp. 41-50). New York, NY: ACM Press.

Olson, L. E., Gunter, C. A., & Madhusudan, P. (2008). A formal framework for reflective database access control policies. In *Proceedings of the 15th ACM Conference on Computer and Communications Security* (pp. 289-298). New York, NY: ACM.

Oracle Corporation (2009). *Oracle® label security with Oracle database 11g release 2*. Retrieved from http://www.oracle.com/technology/deploy /security/database-security/pdf/owp-security-label-security-11gr2.pdf

Oracle Corporation (2010). Oracle® database security guide 11g release 1. Retrieved from http://download.oracle.com/docs/cd/B28359_01/network.111/b28531.pdf

Padma, J., Silva, Y. N., Arshad, M. U., Aref, W. G. (2009). Hippocratic PostgreSQL. In *Proceedings of the 2009 IEEE International Conference on Data Engineering* (pp.1555-1558). Washington, DC: IEEE Computer Society.

Pan, L. (2009). A unified network security and fine-grained database access control model. In *Proceedings of the 2009 Second International Symposium on Electronic Commerce and Security* (pp. 265-269). Washington, DC: IEEE Computer Society.

PHARM (n.d.). *TPC-W Java source code*. Retrieved from http://www.ece.wisc.edu/~pharm/tpcw/tpcw.tar.gz

Pun, S., Chinaei, A. H., & Barker, K. (2009). Twins (1): extending SQL to support corporation privacy policies in social networks. In *Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining* (pp. 306-311). Washington, DC: IEEE Computer Society.

Purevjii, B.-O., Aritsugi, M., Imai, S., & Kanamori, Y. (2007). An implementation design of a fine-grained database access control policy consistency checking mechanism. *Lecture Notes in Computer Science*, 4693, 752-760.

Rask, A., Rubin, D., & Neumann, B. (2005, September 1). *Implementing row and cell-level security in classified databases using SQL Server*. Retrieved from http://technet.microsoft.com/en-us/library/cc966395.aspx

Rastogi, V., Suciu, D., & Welbourne, E. (2008). Access control over uncertain data. *Proceedings of the VLDB Endowment 1*(1), 821-832.

Reddy, N., & Haritsa, J. R. (2005). Analyzing plan diagrams of database query optimizers. In Proceedings of the 31st International Conference on Very Large Data Bases (pp. 1228-1239). New York, NY: ACM Press.

Reed, A. G. (2006). *DeWitt clauses: Can we protect purchasers without hurting Microsoft?* Retrieved from http://www.allbusiness.com /technology/computer-software/4092403-1.html

Rizvi, S., Mendelzon, A., Sudarshan, S., & Roy, P. (2004). Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (pp. 551-562). New York, NY: ACM Press.

Robbert, M. A., & Ricardo, C. M. (2003). Trends in the evolution of the database curriculum. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 139-143). New York, NY: ACM Press.

Roichman, A., & Gudes, E. (2007). Fine-grained access control to web databases. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies* (pp. 31-40). New York: ACM Press.

Rosenthal, A., Sciore, E., & Doshi, V. (1999). Security administration for federations, warehouses, and other derived data. In *Proceedings of the IFIP WG 11.3 Thirteenth International Conference on Database Security: Research Advances in Database and Information Systems Security* (pp. 208-223). Deventer, The Netherlands: Kluwer, B.V.

Rosenthal, A., Dittrich, K., Donahue, J., & Maimone, B. (2001). Will database researchers have any role in data security? In *Proceedings of the 2001 ACM SIGMOD*

International Conference on Management of Data (pp. 621). New York, NY: ACM Press.

Rosenthal, A., & Winslett, M. (2004). Security of shared data in large systems: State of the art and research directions. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (pp. 962-964). New York, NY: ACM Press.

Qamar, N., Ledru, Y., & Idani, A. (2011). Validation of security-design models using Z. In *Proceedings of the 13th International Conference on Formal Methods and Software Engineering* (pp. 259-274). Berlin: Springer-Velag.

Salkind, N. J. (2006). *Exploring Research* (6th ed.). New Saddle River, NJ: Pearson Prentice Hall.

Santos, R. J., & Bernardino, J. (2009). Optimizing data warehouse loading procedures for enabling useful-time data warehousing. In *Proceedings of the 2009 International Database Engineering & Applications Symposium* (pp. 292-299). New York, NY: ACM.

Schroeder, B., Wierman, A., & Harchol-Balter, M. (2006). Open versus closed: A cautionary tale. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation* (pp. 239-252). Berkeley, CA: USENIX Association.

Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., & Price, T. G. (1979). Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data* (pp. 23-34). New York, NY: ACM Press.

Sekaran, U. (2003). *Research methods for business: A skill building approach* (4th ed.). Hoboken, NJ: John Wiley & Sons, Inc.

Shi, J., Zhu, H., Fu, G., & Jiang, T. (2009). On the soundness property for SQL queries of fine-grained access control in DBMSs. In *Proceedings of the 2009 Eighth IEEE/ACIS International Conference on Computer and Information Science* (pp. 469-474). Washington, DC: IEEE Computer Society.

Siegenthaler, M., & Birman, K. (2009). Sharing private information across distributed databases. In *Proceedings of the 2009 Eighth IEEE International Symposium on Network Computing and Applications* (pp. 82-89). Washington, DC: IEEE Computer Society.

Simpson, A. (2008). On the need for user-defined fine-grained access control policies for social networking applications. In *Proceedings of the Workshop on Security in Opportunistic and Social Networks* (pp. 1-8). New York: ACM Press.

Slaymaker, M., Power, D., Russell, D., & Simpson, A. (2008). On the facilitation of fine-grained access to distributed healthcare data. *Lecture Notes in Computer Science, 5159*, 169-184.

Smith, Greg (2007). *Inside the PostgreSQL shared buffer cache*. Retrieved from http://www.westnet.com/~gsmith/content/postgresql/InsideBufferCache.pdf

Steele, R., Gardner, W., & Dillon, T. (2007). XML repository searcher-browser supporting fine-grained access control. *International Journal of Computers & Applications*, *29*(1), 44-50.

Stonebraker, M., & Wong, E. (1974). Access control in a relational data base management system by query modification. In *Proceedings of the 1974 Annual Conference* (pp.180-186). New York, NY: ACM Press.

Stonebraker, M., & Rowe, L. A. (1986). The design of POSTGRESQL. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data* (pp. 340 - 355). New York, NY: ACM Press.

Stonebraker, M. (2008). One size fits all: an idea whose time has come and gone. *Communications of the ACM*, *51*(12), 76-76.

Tjan, H. C. (2006). A combined fine-grained and role-based access control mechanism. *Masters Abstracts International, 44* (5). (UMI No. AAT 1433573) Retrieved June 26, 2007, from Digital Dissertations database

Tolone, W., Ahn, G.-J., Pai, T., & Hong, S.-P. (2005). Access control in collaborative systems. *ACM Computing Surveys*, *37*(1), 29-41.

Transaction Processing Performance Council (1998). *History and overview of the TPC*. Retrieved from http://tpc.org/information/about/history.asp

Transaction Processing Performance Council (2003, February 19). *TPC benchmark W*. Retrieved from http://www.tpc.org/tpcw/spec/TPCWV2.pdf

Transaction Processing Performance Council (2010). *TPC Benchmarks*. Retrieved from http://tpc.org/information/benchmarks.asp

Transaction Processing Performance Council (n.d.). *TPC-W*. Retrieved from http://www.tpc.org/tpcw/default.asp

Turbyfill, C., Orji, C., & Bitton D. (1992). An ANSI SQL standard scalable and portable benchmark for relational database systems. In J. Gray (Ed.), *The Benchmark Handbook* (pp. 317-358). San Mateo, CA: Morgan Kaufmann.

Vieira, M., & Madeira, H. (2005). Towards a security benchmark for database management systems. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks* (pp. 592-601). Washington, DC: IEEE Computer Society.

Wang, G., Liu, Q., & Wu, J. (2010). Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (pp. 735-737). New York, NY: ACM Press.

Wang, Q., Yu, T., Li, N., Lobo, J., Bertino, E., Irwin, K., et al. (2007). On the correctness criteria of fine-grained access control in relational databases. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (pp. 555-566). Berlin: Springer.

Wilson, R. L., & Rosen, P.A. (2003). Protecting data through perturbation techniques: The impact of knowledge discovery in databases. *Journal of Database Management 14*(2): 14-26.

Yu, Y. (2009). Researches on integrating database access control and privacy protection. In *Proceedings of the 2009 Fifth International Conference on Information Assurance and Security* (pp. 330-333). Washington, DC: IEEE Computer Society.

Zhao, F., Nishide, T., & Sakurai, K. (2011). Realizing fine-grained and flexible access control to outsourced data with attribute-based cryptosystems. In *Proceedings of the 7th International Conference on Information Security Practice and Experience* (pp. 83-97). Berlin: Springer-Verlag.

Zhan, G., Li, Z., Ye, X., & Wang, J. (2006). FGAC-QD: Fine-grained access control model based on query decomposition strategy. *Lecture Notes in Computer Science, 4083*, 132-141.

Zhang, H. (2008). Query evaluation in the presence of fine-grained access control. *Dissertation Abstracts International, 69* (11), 2474B. (UMI No. 3133261) Retrieved February 28, 2009, from Digital Dissertations database.

Zhang, H., Ilyas, I. F., & Salem, K. (2009). PSALM: Cardinality estimation in the presence of fine-grained access controls. In *Proceedings of the 2009 IEEE International Conference on Data Engineering* (pp. 501-506). Washington, DC: IEEE Computer Society.

Zhu, H., Xin, F., Hui, L. Q., & Lu, K. (2006). The design and implementation of a performance evaluation tool with TPC-W benchmark. *Journal of Computing & Information Technology, 14*(2), 149-159.

Zhu, H., & Lü, K. (2007). Fine-grained access control for database management systems. *Lecture Notes in Computer Science, 4587*, 215-223.

Zhu, H., Shi, J., Wang, Y., & Feng, Y. (2008). Controlling information leakage of fine-grained access model in DBMSs. In *Proceedings of the Ninth International Conference on Web-age Information Management* (pp. 583-590). Washington, DC: IEEE Computer Society.

The Graduate School of Computer and Information Sciences

Certification of Authorship of Dissertation Work

Submitted to (Advisor's Name): Dr. Junping Sun

Student's Name: David H. Kumka

Date of Submission: October 28, 2012

Purpose and Title of Submission: Quantifying Performance Costs of Database Fine-Grained Access Control

Certification of Authorship: I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas, or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this paper was prepared by me for this purpose.

Student's Signature: David H. Kumka_____